Conflict-driven constraint answer set solving

Author:
Drescher, Christian

Publication Date:
2015

DOI:
https://doi.org/10.26190/unsworks/18170

Conflict-driven Constraint Answer Set Solving

Christian Drescher

A thesis in fulfilment of the requirements for the degree of
Doctor of Philosophy

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

School of Computer Science and Engineering
Faculty of Engineering

May 2015

Abstract

Constraint answer set programming (CASP) is a declarative problem solving paradigm that combines the strengths of constraint programming (CP) and answer set programming (ASP). ASP solvers provide good computational performance by conflict-driven nogood learning (CDNL) and exploiting unfounded sets. To model features like finite domain variables and global constraints, which are naturally dealt with in CP, hybrid CASP systems have been developed that delegate CP constructs to a CP solver. While this achieved some success, hybrid systems do not seamlessly integrate with CDNL. We address this deficiency by devising two alternative approaches that accommodate CDNL.

For one, we introduce a translation-based approach. The idea is to enhance ASP with CP constructs through translation to ASP. Implemented as preprocessing, this allows us to apply existing CDNL-enabled ASP systems to CASP solving. Our contributions include various generic translations that work for any constraint, and specialised encodings for important global constraints such as ALL-DIFFERENT, GRAMMAR, and REACHABILITY. We show that the inference of ASP solvers can simulate the effect of complex CP algorithms in many cases. Propagation of REACHABILITY, however, can be hindered because ASP systems disregard some consequences from unfounded sets for performance reasons. We tackle this weakness by providing more efficient methods using a reduction to the problem of finding dominators in a flowgraph.

For another, we devise an extension to CDNL-based ASP solving that can integrate CP constructs via lazy nogood generation (LNG). Rather than a-priori translations into ASP, the idea of LNG is to make necessary parts of an encoding explicit on demand and only when new information can be propagated. We introduce external propagators to facilitate LNG and incorporate them into a decision procedure for ASP solving that is centred around CDNL. We then demonstrate how to seamlessly integrate constraint propagation with this framework, resulting in a novel approach to CASP solving.

We have implemented a prototypical CASP system to demonstrate some key principles of our approaches. In 2013, it has successfully participated in a model and solve competition, outperforming hybrid systems.

Acknowledgements

This PhD thesis is the culmination of a life-long interest in the area of artificial intelligence that was fuelled by so many people. Accomplishing this thesis would not have been possible without them.

I am most grateful to Toby Walsh for supervising my PhD project. His academic courage and vision inspires me. I thank him for providing me guidance, many constructive comments, and putting in all his weight into winning me scholarships and travel grants. I also thank my co-supervisor Maurice Pagnucco for his support and help.

I am indebted to Torsten Schaub for introducing me into research during my earlier studies. Special thanks go to him and Tomi Janhunen for taking on the task of reviewing this thesis. I am grateful to Pascal van Hentenryck and Peter Stuckey for invaluable discussions on various occasions. I also thank Marcello Balduccini, Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Yuliya Lierler, Valentin Mayer-Eichberger, Nina Narodytska, and Max Ostrowski for their brilliant work.

Finally, and most importantly, I thank my wife Nino Labadze, my family, and friends for supporting me all the time.

Contents

Chapter 1

Introduction

Developing a powerful paradigm for declarative problem solving is one of the key challenges in the area of knowledge representation and reasoning. A promising candidate is answer set programming (ASP; Gelfond and Lifschitz, 1988; Eiter et al., 1997; Lifschitz, 1999; Niemelä, 1999; Marek and Truszczyński, 1999; Baral, 2003; Gebser et al., 2012b) that has emerged in the last decade of the 20th century. It comes with an expressive but simple modelling language rooted in logic programming (Baral and Gelfond, 1994) and nonmonotonic reasoning (Brewka et al., 2008), featuring recursive definitions and default negation. In fact, the integration with conflict-driven solving (Mitchell, 2005) and the exploitation of unfounded sets (Van Gelder et al., 1991) in modern ASP solvers (Giunchiglia et al., 2006; Gebser et al., 2007a; Maratea et al., 2008; Lierler, 2011; Alviano et al., 2013b) achieved a breakthrough in terms of computational performance that has elevated ASP to the avant-garde in logic programming.

ASP now faces a growing demand for new modelling features, most notably variables over finite domains and global constraints, constructs that are more naturally handled in constraint programming (CP; Dechter, 2003; Rossi et al., 2006). As demonstrated in (Dovier et al., 2005; Coban et al., 2008; Mancini et al., 2008; Celik et al., 2009), ASP has the advantage of being more compact, more elaboration tolerant than CP, and is highly competitive. On the other hand, it sometimes makes modelling difficult and tedious, and produces non-compact ASP encodings, whilst in CP, finite domain variables facilitate elegant and general modelling, and a large library exists for sophisticated propagation algorithms of global constraints. On the other hand, CP implementations are less robust and cannot handle recursive definitions naturally.

This led to the development of constraint answer set programming (CASP; Baselice et al., 2005), a paradigm that naturally merges CP and ASP, while preserving the advantages of both approaches. For illustration, consider the problem of solving a Hidato puzzle.

Example: CASP Model of Hidato

Hidato (Benedek, 2008) is a number-placement puzzle game. The goal is to fill a grid with consecutive integers connecting horizontally, vertically, or diagonally. Hence, all values assigned to the cells in a solution must be all-different. Some numbers, including 1, being the smallest, and the highest number n on the grid, are preassigned in order to guarantee a unique solution. The following is a sample Hidato containing 80 cells.

						28	26		
					30				
	66		37			16			
71		63		36	32		18		22
	70		41	35		14		11	21
		44		42				8	
	60		46	48			9		7
				49		51		1	
		79			50				3
			80						

Every Hidato puzzle can be described in terms of logic programming facts, where

- connections between cells are encoded in the form `edge(X,Y)`, representing that a cell `X` is connected to `Y`
- preassigned values are encoded in the form `clue(X,V)`, representing that a cell `X` takes the value `V`, and
- the highest integer `N` on the grid, i.e., the number of cells in the grid, is encoded in the form `size(N)`.

Then, the above instance is represented by the following:

```
edge(cell_0_1,cell_0_2).
edge(cell_0_2,cell_0_1).
```

```
edge(cell_0_1,cell_1_1).
edge(cell_1_1,cell_0_1).
...
clue(cell_0_6,28).
clue(cell_0_7,26).
clue(cell_2_2,66).
...
size(80).
```

Logic programming based approaches like ASP are particularly well suited for modelling Hidato because path constraints are encoded straightforwardly using recursive transitive closure. This is difficult to encode in CP. On the other hand, constraints that work globally, like ALL-DIFFERENT on the values taken by the cells, are more naturally represented in CP.

The purpose and strength of CASP is that it combines the best of both worlds. We demonstrate CASP using the syntax of the CASP system *inca* [1].

```
#var $value(X) : cell(X) : size(N) = 1..N.                (1.1)
$value(X) #== V :- clue(X,V).                             (1.2)
#alldifferent {$value(X) : cell(X)}.                      (1.3)
1 {path(X,Y) : edge(X,Y)} 1 :- cell(Y), not clue(Y,1).    (1.4)
reached(X) :- clue(X,1).                                  (1.5)
reached(Y) :- reached(X), path(X,Y).                      (1.6)
:- cell(X), not reached(X).                               (1.7)
#linear [$value(Y), -$value(X)] == 1 :- path(X,Y).        (1.8)
```

The encoding works on a description of the puzzle in terms of logic programming facts. As you can see, CASP distinguishes regular atoms from constraint atoms and variable declarations, indicated by the `#` symbol, and constraint variables, indicated by the `$` symbol. Line (1.1) declares a variable for each cell in the grid with values between 1 and `N`, whilst Line (1.2) ensures that pre-assignments are respected by the corresponding variables. Line (1.3) encodes the condition that the values taken by the cells in the grid are ALL-DIFFERENT. Lines (1.4–1.7) encode a Hamiltonian path, represented by auxiliary atoms

of the form `path(X,Y)`, and Line (1.8) ensures that the values taken by two connected cells in the Hamiltonian path are consecutive. Note that the ALL-DIFFERENT constraint is redundant in this encoding, but it can greatly improve run time.

						28	26		
67	65			38	30	29	27	25	24
68	66	64	37	39	31	16	17	19	23
71	69	63	40	36	32	15	18	20	22
72	70	62	41	35	33	14	12	11	21
73	61	44	43	42	34	13	10	8	6
74	60	45	46	48	54	53	9	5	7
75	59	58	47	49	55	51	52	1	4
76	77	79	57	56	50			2	3
		78	80						

Above graphic shows the unique solution of the sample Hidato puzzle.

Although CASP systems support models with first-order variables, this thesis limits the attention to propositional specifications and the process of constraint answer set solving.

At the solving level, recent years have seen the emergence of *hybrid* CASP systems (Mellarkod and Gelfond, 2008; Mellarkod et al., 2008; Balduccini, 2009; Gebser et al., 2009c). Following the idea of satisfiability modulo theories (SMT; Nieuwenhuis et al., 2006), hybrid CASP systems improve on the solving capabilities of ASP by delegating the tasks of handling finite domain variables and the propagation of constraints to CP systems. This approach has been successfully applied to sophisticated industrial-size scheduling problems (Balduccini, 2011), and comparisons to the state-of-the-art in ASP are encouraging (Balduccini and Lierler, 2012; Lierler et al., 2012). As demonstrated by Gebser et al. (2009c), the key to developing an efficient CASP system is the integration with conflict-driven solving, most importantly CONFLICTANALYSIS (Moskewicz et al., 2001), i.e., capable of combining conflict information from ASP and CP constructs.

1.1 Contributions

Although hybrid CASP systems have achieved some success, their design principle does not integrate well with the state-of-the-art in conflict-driven solving: Al-

[1] `http://potassco.sourceforge.net/labs.html`

though modern hybrid CASP systems Gebser et al. (2009c); Balduccini (2009) employ a conflict-driven ASP solver as their core reasoning engine, enabling learning, conflict-oriented search heuristics, and conflict-directed backjumping (Mitchell, 2005), their performance can be hindered by the CP counterpart, as CP solvers typically apply backtracking search (Dechter, 2003; Rossi et al., 2006). The major limiting factor, however, is the lack of information-sharing between the ASP and CP systems. In particular, the elaboration of constraint interdependencies through CONFLICTANALYSIS is restricted because current CP systems do not provide an interface supporting conflict-driven learning.

We address these deficiencies by devising two alternative approaches to constraint answer set solving that fully accommodate conflict-driven solving.

Translation-based Constraint Answer Set Solving

We introduce a *translation-based* approach to CASP solving rather than a hybrid one. The idea is to enhance ASP with CP techniques through translation to ASP. In other words, any CASP specification is compiled into an ASP encoding by adding a logic programming decomposition of all CP constructs. We present various generic translations that work for any constraint, and specialised encodings for a selection of important global constraints such as ALL-DIFFERENT (van Hoeve, 2001), GRAMMAR and related constraints (Pesant, 2004; Sellmann, 2006; Quimper and Walsh, 2006), and REACHABILITY (Dooms et al., 2005). Implemented as preprocessing, any existing ASP systems can be applied without changing their source code.

While this enables conflict-driven CASP solving, the effect of propagating conditions encoded into ASP is not yet understood. Hence, our major contribution is a study of the interaction between the ASP representation of a constraint and the inferences made by ASP solvers in terms of local consistency (Dechter, 2003; Rossi et al., 2006). In particular, we show that ASP inference on our encodings can simulate the effect of complex CP algorithms in many cases.

An exception is given through REACHABILITY, a condition that should be naturally and efficiently handled in ASP, since ASP allows for recursive transitive closure and its solvers employ very efficient inference mechanisms such as unit propagation (UP) and well-founded negation (WFN). Whilst this intuition is strengthened by experimental results (Celik et al., 2009; Coban et al., 2008), we show that the propagation of REACHABILITY can be hindered because ASP systems, for performance reasons, disregard some consequences from unfounded sets. We tackle

this weakness by providing a more efficient method to compute the effects of well-founded justification (WFJ) and another, new form of unfounded set inference, called well-founded domination (WFD). Using a reduction to the problem of finding dominators in a flowgraph, we show how to approximate the consequences from WFJ and WFD in general, but compute all their effects in some relevant cases, including all uses of REACHABILITY.

Constraint Answer Set Solving via Lazy Nogood Generation

We devise an extension to conflict-driven nogood learning (CDNL; Gebser et al., 2007a) based ASP solving that integrates constraint propagation and the handling of finite domain variables via lazy nogood generation (LNG), a technique that is founded on very recent developments (Ohrimenko et al., 2009) in the area of constraint satisfaction problem (CSP; Dechter, 2003; Rossi et al., 2006) research. Rather than translating CP constructs into ASP a-priori, the central idea of LNG is to encode necessary parts of their translation on demand and only when new information can be propagated. This is motivated by the insight that the performance of ASP systems is sensitive to the size of encodings, which can quickly become impractical.

We here lay the foundation to LNG for ASP in terms of external propagators that represent conditions on the answer sets of a logic program without encoding them a-priori, but that make their encoding explicit when they trigger any inference. We also specify a decision procedure for CASP solving with LNG. It is centred around CDNL and integrates constraint propagation without sacrificing the advantages of conflict-driven techniques. To demonstrate the applicability of our approach, we have implemented a prototypical CASP system. In 2013, it has successfully participated in a competition, outperforming hybrid systems.

Related Publications

The research presented in this thesis has been carried out with the intellectual support of Toby Walsh. The theoretical and experimental contributions are original work by the author.

Some material on the generic *value* and *support* encodings (Sections 4.2–4.3), and the support encoding of the ALL-DIFFERENT constraint (Definition 5.2; Section 5.1) has been presented in the preliminaries of a previous Master's thesis (Drescher, 2010), but was not claimed as a contribution of it.

Part of the results presented in this thesis have been reported in the following publications.

C. Drescher and T. Walsh. Efficient approximation of well-founded justification and well-founded domination. In *Proceedings of LPNMR'13*, pages 277–289. Springer, 2013.

C. Drescher and T. Walsh. Answer set solving with lazy nogood generation. In *ICLP'12 Technical Communications*, pages 188–200. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2012.

C. Drescher and T. Walsh. Conflict-driven constraint answer set solving with lazy nogood generation. In *Proceedings of AAAI'11*, pages 1772–1773. AAAI Press, 2011.

C. Drescher and T. Walsh. Modelling grammar constraints with answer set programming. In *ICLP'11 Technical Communications*, pages 28–39. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2011.

C. Drescher and T. Walsh. Translation-based constraint answer set solving. In *Proceedings of IJCAI'11*, pages 2596–2601. IJCAI/AAAI Press, 2011.

C. Drescher and T. Walsh. A translational approach to constraint answer set solving. *Theory and Practice of Logic Programming*, 10(4-6):465–480, 2010.

C. Drescher and T. Walsh. Reformulation of global constraints into answer set programming. In *Proceedings of AAAI Workshop WARA'10*, pages 14–19. AAAI Press, 2010.

C. Drescher. Constraint answer set programming systems. In *ICLP'10 Technical Communications*, pages 255–264. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

1.2 Outline

The remainder of this thesis is organised as follows.

In Chapter 2, we presents all necessary background, starting with notions from graph theory and formal language theory. Then, we proceed with concepts from CSP and Boolean satisfiability (SAT). A particular focus is put on UP and CDNL. We then introduce ASP and a characterisation of the conditions to the answer sets

of a logic program in terms of nogoods and unfounded sets. Given this, we present important preliminaries to unfounded set inference, in particular, WFN and WFJ, and two nogood-based alternatives. Then, we provide an introduction to important extensions to the language of ASP, and a central theorem used in this thesis, i.e., the splitting set theorem (Lifschitz and Turner, 1994) based on logic programs with externals. Finally, we acquaint the reader with CASP and outline the state-of-the-art in constraint answer set solving.

In Chapter 3, we provide our method for approximating the consequences of WFJ, WFD, and related inference. We begin with an introduction to support flow-graphs, our new graph-representation of programs written in ASP, and show how the effect of WFJ can be approximated via cuts in the graph. Based on this, we prove that the problem of finding all dominators in such graph can be used to approximate WFJ. We also demonstrate that its effects can be simulated for important classes of programs. Then, we introduce WFD and show how our method can be used to approximate, respectively simulate its consequences too. We conclude the chapter with a discussion of the limitations of our approach as well as its related work.

In Chapter 4, we describe our translational approach to constraint answer set solving. First, we present the foundations of our key idea, i.e., compiling all CP constructs from a CASP encoding into ASP by exploiting Lifschitz and Turner's splitting set theorem. Then, we present how the domain of a finite domain variable can be represented with ASP in various ways. We begin with the value encoding. Based on this, we demonstrate two alternative generic encodings of constraints and compare the local consistency achieved by UP. We proceed with a bound and a range representation of the variables' domains in a similar fashion, and outline mixed variants. Finally, we discuss related work.

In Chapter 5, we proceed with our translational approach to constraint answer set solving by considering specialised encodings of important global constraints. We begin with ALL-DIFFERENT for which we present encodings such that the inference of any modern ASP solver achieves arc, bound, and range consistency. For each case, we prove its asymptotic run time complexity. We give some experimental results demonstrating competitiveness with related work. We then proceed with two alternative encodings of GRAMMAR and related constraints. One of which is more straightforward, with the other being an extension such that ASP inference maintains domain consistency. We prove that our models exhibit an asymptotic run time complexity that is similar to the one of complex propagation algorithms.

Then, we proceed with the REGULAR constraint in a similar fashion. Experimental results demonstrate competitiveness with related work. A final, global constraint considered in this thesis is REACHABILITY. We first show how the domains of graph variables and vertex set variables can be represented with ASP. Then, we proceed with studying the impact of ASP inference on propagating REACHABILITY with varying degrees on freedom. Our results will establish practical relevance for WFJ and WFD inference. Again, we prove asymptotic run time complexity and present the outcome of an experimental evaluation. We conclude the chapter with a discussion of the limitations of our translation-based approach to constraint answer set solving as well as its related work.

In Chapter 6, we describe our second approach to constraint answer set solving which is facilitated through LNG. We begin with presenting its logical foundations by formulating a variant of the splitting set theorem that considers external propagation. We then demonstrate how constraint propagation integrates with our framework. Based on this, we present a decision procedure that is centred around conflict-driven solving. We empirically evaluate our approach and compare to the state-of-the-art in ASP and CASP. Finally, we discuss related work.

In Chapter 7, we conclude the main part of this thesis, and outline the significance of the presented results.

Chapter 2

Background

We start by recalling some basic notions of formal graph and language theory, and give all necessary background to constraint satisfaction problems, answer set programming, and constraint answer set programming.

2.1 Graph Theory

Many structures are conveniently represented by means of a graph, including, as we shall see, structures in answer set programming. In this thesis, we will mostly consider finite, directed graphs.

A *(directed) graph* $\mathsf{G} = (V, E)$ consists of a finite set V or $V(\mathsf{G})$ of *vertices* and a set of ordered pairs of distinct vertices E or $E(\mathsf{G})$ called *(directed) edges*. If v and w are two vertices of a graph and the ordered pair $e = (v, w)$ is an edge of the graph, we say that e is *directed from v to w*, that e is *adjacent from v*, and that e is *adjacent to w*. The size of a graph G is the number of its edges, i.e., $|E(\mathsf{G})|$.

An *undirected graph* is one in which edges are unordered pairs, i.e., they have no orientation. In other words, the edge (v, w) is identical to the edge (w, v). Graph inclusion defines a partial ordering among graphs. For two graphs G and G', G *includes* G', denoted by $\mathsf{G} \subseteq \mathsf{G}'$, if $V(\mathsf{G}) \subseteq V(\mathsf{G}')$ and $E(\mathsf{G}) \subseteq E(\mathsf{G}')$. We also say that G' is a *sub-graph* of G.

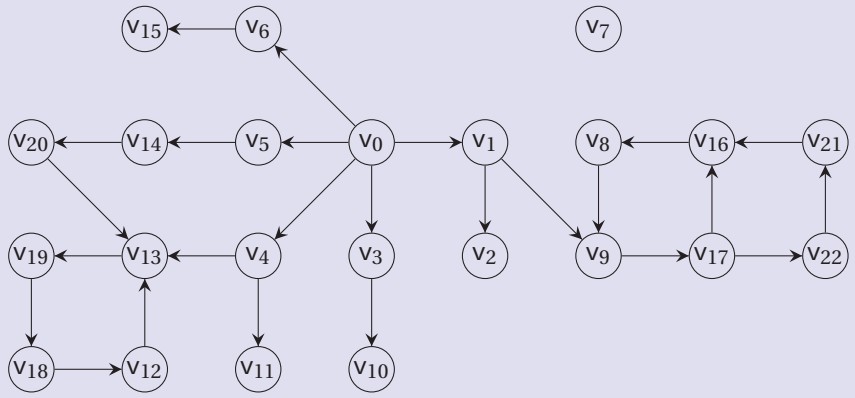
We illustrate graphs as diagrams in which the vertices are represented by circles and boxes with textual labels and the edges are represented by arrows.

Example 2.1

The following is an illustration of the graph $\mathsf{G} = (V, E)$ where $V = \{v_0, \ldots, v_{22}\}$,

and

$$E = \left\{ \begin{array}{l} (v_0, v_1), (v_0, v_3), (v_0, v_4), (v_0, v_5), (v_0, v_6), (v_1, v_2), (v_1, v_9), \\ (v_3, v_{10}), (v_4, v_{11}), (v_4, v_{13}), (v_5, v_{14}), (v_6, v_{15}), (v_8, v_9), (v_9, v_{17}), \\ (v_{12}, v_{13}), (v_{13}, v_{19}), (v_{14}, v_{20}), (v_{16}, v_8), (v_{17}, v_{16}), (v_{17}, v_{22}), \\ (v_{18}, v_{12}), (v_{19}, v_{18}), (v_{20}, v_{13}), (v_{21}, v_{16}), (v_{22}, v_{21}) \end{array} \right\}.$$



Connectivity

Let v and w be two vertices of a graph G. A *path* from v to w is a finite alternating sequence $(\mathsf{v}_0, e_1, \mathsf{v}_1, e_2, \ldots, e_n, \mathsf{v}_n)$ of vertices and edges of the graph such that $v = \mathsf{v}_0$, $w = \mathsf{v}_n$, and $e_i = (\mathsf{v}_{i-1}, \mathsf{v}_i)$ for $1 \leq i \leq n$. If u is a vertex in the path then the path is said to *pass through* u. It is not required that the vertices in a path are all distinct, i.e., a path can pass through a vertex more than once. The number of edges in a path is the *length* of the path. If there is a path from v to w, then v *connects to* w, and we also say that w is *connected to* v. Otherwise, w is *disconnected from* v. A graph is *connected* if for any two vertices v and w of the graph either v connects to w or w connects to v. A connected sub-graph G' of a graph G is a *loop* if for any two vertices v and w of the graph v connects to w and w connects to v. If, in addition, every graph G'' such that $\mathsf{G}' \subset \mathsf{G}'' \subseteq \mathsf{G}$ is not a loop, then G' is a strongly connected component (SCC) of G. In other words, an SCC of a graph is a maximal loop. Tarjan (1972) proposed a linear-time algorithm for finding all SCCs in a graph.

Example 2.2

Reconsider the graph G from Example 2.1. The sequences s_0, and s_1, given

through

$$s_0 = (v_0, (v_0, v_4), v_4, (v_4, v_{13}), v_{13}, (v_{13}, v_{19}), v_{19}, (v_{19}, v_{18}), v_{18}), \text{ and}$$
$$s_1 = (v_0, (v_0, v_5), v_5, (v_5, v_{14}), v_{14}, (v_{14}, v_{20}), v_{20}, (v_{20}, v_{13}), v_{13},$$
$$(v_{13}, v_{19}), v_{19}, (v_{19}, v_{18}), v_{18})$$

both are paths from v_0 to v_{18}. Verify that the graph $\mathsf{G}' = (V', E')$, where

$$V' = \{v_{12}, v_{13}, v_{18}, v_{19}\} \text{ and}$$
$$E' = \{(v_{12}, v_{13}), (v_{13}, v_{19}), (v_{19}, v_{18}), (v_{18}, v_{12})\},$$

is a loop of G. On the other hand, v_7 is disconnected from any other vertex. In particular, G is not a connected graph.

We will also consider trees, that are, a special case of graphs. A *(directed rooted) tree* is a connected graph with a unique vertex, the *root*, and in which for any other vertex v, there is exactly one path from the root to v. In a tree, any vertex w that has an edge adjacent from another vertex v is called a *child* of v.

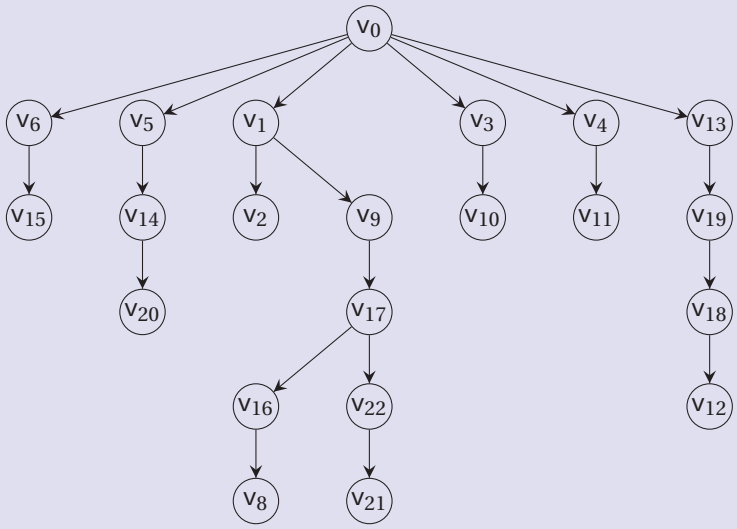
Cuts and Dominators in Flowgraphs

In our context, a *flowgraph* is a graph G with a specially designated *source* vertex. A vertex v of a flowgraph G is *reachable* if v is connected to the source. If every path from the source to v passes through a vertex w, then w *dominates* v. Note that, by definition, every vertex dominates itself. If w dominates v and $w \neq v$ then w *strictly dominates* v. If w strictly dominates v and there is no other vertex u such that u strictly dominates w and w strictly dominates u, then w *immediately dominates* v. Similarly, we define for a set $W \subseteq V$ of vertices, w *dominates* W if w dominates every $v \in W$, and w *immediately dominates* W if w dominates W and there is no vertex u such that u dominates W and w strictly dominates u.

The domination relationship induced by a flowgraph G can be represented in form of a graph, where each edge directs a vertex to those vertices it immediately dominates. If G is connected then every vertex is immediately dominated by at most one other vertex. The resulting graph is a tree with the source vertex as root, and is therefore called the *dominator tree*. In our context, we will disregard all vertices that are disconnected from the source when illustrating the domination relationship of a graph. Thus, the domination relationship of any flowgraph is a dominator tree.

Example 2.3

Reconsider the graphs from Example 2.2. Verify that G is a flowgraph with source v_0. The dominator tree of G is illustrated below.



Observe that, although the sub-graph G′ forms a loop of G, its vertices are in a strict domination relationship with v_{13} immediately dominating $\{v_{12}, v_{18}, v_{19}\}$, v_{12} immediately dominating $\{v_{18}, v_{19}\}$, and v_{18} immediately dominating v_{19}.

Georgiadis and Tarjan (2004) have provided a linear-time algorithm for finding all dominators in a flowgraph. It can be made incremental, i.e., few dominators might be recomputed at any stage during search, subject to removal and re-insertion of edges (Sreedhar et al., 1997).

Next, consider any partition $\mathsf{c} = (S, W)$ of the vertices $V(\mathsf{G})$ into two disjoint subsets S and W such that the source is in S. For accessing the vertices in S that have an edge directed to a vertex in W, define

$$\text{front}(\mathsf{c}) = \{v \in S \mid (v, w) \in E(\mathsf{G}), w \in W\}.$$

In principle, edges that are directed from a vertex in W to a vertex in S are allowed. For vertices in W that have an edge directed to a vertex in S, define

$$\text{back}(\mathsf{c}) = \{v \in W \mid (v, w) \in E(\mathsf{G}), w \in S\}.$$

The partition c is called a *cut* in G since no vertex in W is reachable if all edges adjacent from vertices in front(c) are removed from $E(\mathsf{G})$. It is easy to see that, front(c) $= \{v\}$ if and only if v immediately dominates W.

Example 2.4

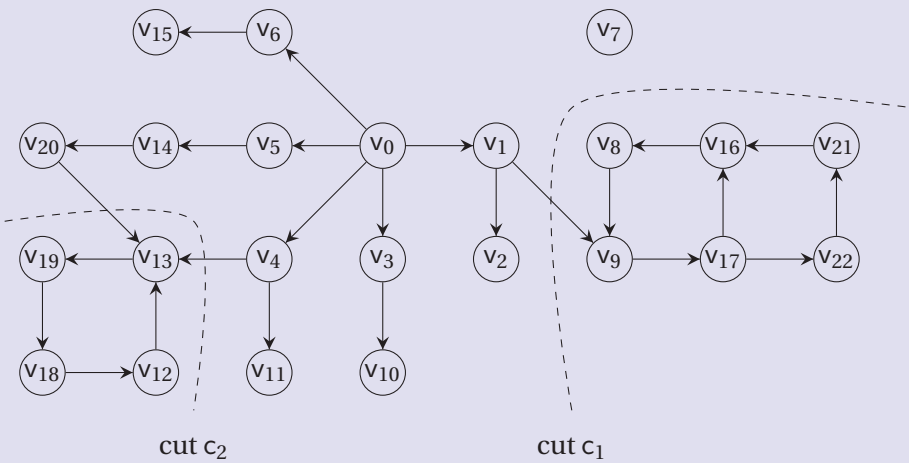
Reconsider the graph from Example 2.3. Consider the cuts c_1 and c_2 given through

$$c_1 = (\{v_0, \ldots, v_7, v_{10}, \ldots, v_{15}, v_{18}, v_{19}, v_{20}\}, \{v_8, v_9, v_{16}, v_{17}, v_{21}, v_{22}\}), \text{ and}$$
$$c_2 = (\{v_0, \ldots, v_{11}, v_{14}, \ldots, v_{17}, v_{20}, v_{21}, v_{22}\}, \{v_{12}, v_{13}, v_{18}, v_{19}\}).$$

Verify that c_1 and c_2 are in fact cuts of G with

$$\text{front}(c_1) = \{v_1\}, \text{ and } \text{front}(c_2) = \{v_4, v_{20}\}.$$

Also note that $\text{back}(c_1) = \emptyset$ and $\text{back}(c_2) = \emptyset$. The cuts are illustrated in the diagram below.



Verify that the vertex $\text{front}(c_1) = \{v_1\}$ immediately dominates the set of vertices $\{v_8, v_9, v_{16}, v_{17}, v_{21}, v_{22}\}$. On the other hand, neither of the vertices from $\text{front}(c_2) = \{v_4, v_{20}\}$ immediately dominates any of the vertices in the loop G' as there are two alternative paths to reach the vertices in the loop, that are, one that passes through v_4 and one that passes through v_{20}.

2.2 Formal Languages

We now give the necessary background from formal language theory.

A *string* is a sequence of *symbols* from an alphabet Σ. The *length* of a string is given through the length of the sequence, i.e., the number of indices for symbols in

the string. The set of all strings over Σ is denoted by Σ^*. Then, a *formal language* L is a subset of Σ^*. Among other formalisms, formal languages are usually described by formal grammars or automata.

Grammars

We will consider context-free, linear, and regular grammars. A context-free grammar (CFG; Chomsky, 1956) is a quadruple $\mathscr{G} = (N, \Sigma, P, S)$, where N is a finite set of *nonterminal* symbols, Σ is a finite set of *terminal* symbols (the alphabet, being disjoint from N), $P \subseteq N \times (N \cup \Sigma)^*$ is a set of *production* rules, and the distinguished *start* symbol $S \in N$. The grammar $\mathscr{G}$ is *linear* if $P \subseteq N \times \Sigma^* \cup N^*\Sigma^*N^*$, and *regular* if $P \subseteq N \times \Sigma \cup \Sigma N$. Hence, regular grammars are strictly contained in linear grammars, and linear grammars are strictly contained in CFG.

We often omit to specify the complete quadruple and only provide the set of production rules using the following conventions in the context of grammars: capital letters denote nonterminals in N, lowercase letters denote terminal symbols in Σ, and v and ω (sometimes with index) denote a sequence of nonterminals and terminals called *string*. We also assume that S is the unique start symbol. Moreover, productions $(A, \omega) \in P$ can also be written as $A ::= \omega$, and productions $(A, \omega_1), (A, \omega_2), \ldots, (A, \omega_m) \in P$ can be written as $A ::= \omega_1 \mid \omega_2 \mid \cdots \mid \omega_m$. We define the size of the grammar $|\mathscr{G}|$ as the number of productions in P.

The derivation relationship $\Rightarrow_{\mathscr{G}}$ induced by the grammar $\mathscr{G}$ is defined for any v_1, v_2 as $v_1 A v_2 \Rightarrow_{\mathscr{G}} v_1 \omega v_2$ if there is a production $A ::= \omega \in P$. We write $\omega_1 \Rightarrow^*_{\mathscr{G}} \omega_m$ if there exists a sequence of strings $\omega_2, \ldots, \omega_{m-1}$ such that $\omega_i \Rightarrow_{\mathscr{G}} \omega_{i+1}$ for all $1 \leq i < m$. If $\omega_1 \Rightarrow^*_{\mathscr{G}} \omega_m$ then we say that ω_1 *produces* ω_m. The *language produced by* $\mathscr{G}$ is the set of strings $L_{\mathscr{G}} = \{\omega \in \Sigma^* \mid S \Rightarrow^*_{\mathscr{G}} \omega\}$.

A CFG is in *Chomsky normal form* if all productions are of the form $A ::= a$ or $A ::= BC$. Every context-free grammar $\mathscr{G}$ such that the empty string ε is not generated by $\mathscr{G}$ can be transformed into a grammar $\mathscr{H}$ such that $L_{\mathscr{G}} = L_{\mathscr{H}}$ and $\mathscr{H}$ is in Chomsky normal form. Transformations are described in most textbooks on automata theory, such as (Hopcroft and Ullman, 1979), with at most a linear increase in the size of the grammar.

Example 2.5

Consider the following CFG $\mathscr{G}$ given through the productions

$$S ::= SA \mid AS \mid 2$$

$A ::= 1$

Observe that $\mathscr{G}$ is in Chomsky normal form. The grammar produces the language of strings that contain a single 2, preceded or succeeded by a sequence of 1s of any length. For instance, the strings 2, 12, 21, 112, 121, 211, and so on, are in $L_{\mathscr{G}}$. In particular, we derive $S \Rightarrow_{\mathscr{G}} AS \Rightarrow_{\mathscr{G}} 1S \Rightarrow_{\mathscr{G}} 12$.

The Cocke-Younger-Kasami Parsing Algorithm

To determine whether a string is contained in the language generated by a CFG, we consider the Cocke-Younger-Kasami (CYK) parsing algorithm (Younger, 1967). *Parsing* denotes a process of analysing a sequence of symbols.

The CYK algorithm requires the CFG to be rendered into Chomsky normal form and constructs a dynamic programming table T where, considering every possible subsequence of symbols, it includes a nonterminal $A \in N$ into $T[i, j]$ if A produces the string from the i-th symbol of length j. First, the CYK algorithm starts with subsequences of length 1. Then, it goes on to subsequences of length 2 and so on, considering every split of a sequence from the i-th symbol of length j into two subsequences, say of length k and $j-k$ for $1 \leq k < j$, and checks if there is some production $A ::= BC \in P$ such that $B \in T[i, k]$ and $C \in T[i+k, j-k]$. For each such production, the algorithm records $A \in T[i, j]$, meaning that A produces the considered subsequences. Once this process terminates, the inclusion of the start symbol S into $T[1, n]$ determines whether the input string of length n is contained in the language generated by the CFG.

Example 2.6

The dynamic programming table T is typically represented by a grid. For instance, reconsidering the setting from Example 2.5, i.e, parsing the string 12, CFG generates the following table:

{S}	
{A}	{S}
1	2

Now, consider the slightly extended grammar $\mathscr{G}'$ that generates sequences of strings produced from the above grammar, separated by two 1s:

$S ::= SA \mid AS \mid 2 \mid BS$

$A ::= 1$

$B ::= SC$

$C ::= AA$

For the input string 2112, we get the following table:

{S}			
{B, **S**}	{**S**}		
{**S**}	{C}	{**S**}	
{S}	{A}	{A}	{S}
2	1	1	2

Observe that the bold-faced entries are not used in any sequence of derivations that starts with S and produces 2112.

In light of above observation, i.e., the CYK algorithm includes nonterminals in the dynamic programming table even if they are not used in any sequence of derivations that starts with S and produces the input string, we make the following distinction between entries in $T[i,j]$: Let ω be the string of length j from the i-th symbol. We say that $A \in T[i,j]$ *acts in a successful parsing* of the input string $v_1\omega v_2 \in L_{\mathcal{G}}$ if $A \Rightarrow^*_{\mathcal{G}} \omega$ and $S \Rightarrow^*_{\mathcal{G}} v_1 A v_2 \Rightarrow^*_{\mathcal{G}} v_1\omega v_2$.

Example 2.7

Reconsider the dynamic programming table for parsing the string 2112 from Example 2.6. We have that $S \in T[1,1]$, $A \in T[2,1]$, $A \in T[3,1]$, $S \in T[4,1]$, $C \in T[2,2]$, $B \in T[1,3]$, and $S \in T[1,4]$ act in a successful parsing of the input string, whilst $S \in T[1,3]$, $S \in T[2,3]$, $S \in T[1,2]$, $S \in T[3,2]$ do not.

Automata

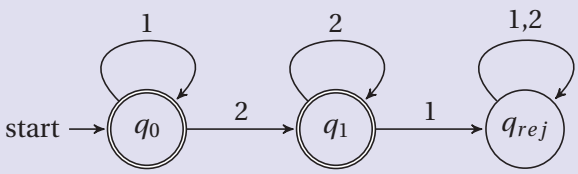
A deterministic finite automaton (DFA; Hopcroft and Ullman, 1979) $\mathcal{M}$ is a quintuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite, non-empty set of *states*, Σ is a finite, non-empty input alphabet, δ is a transition function mapping state-input symbol pairs to a new state $Q \times \Sigma \to Q$, q_0 is the initial state, and F is a set of accepting states.

A DFA takes a sequence of input symbols ω as input, each symbol $t \in \Sigma$ causes $\mathcal{M}$ to perform a transition from its current state q to a new state $\delta(q,t)$, where $\mathcal{M}$ starts off in the state q_0. The input ω is *recognised* by $\mathcal{M}$ if ω causes $\mathcal{M}$ to transition from q_0 in one of the accepting states. The *language recognised by* $\mathcal{M}$ is the set of inputs $L_{\mathcal{M}} = \{\omega \in \Sigma^* \mid \mathcal{M} \text{ recognises } \omega\}$.

DFAs are illustrated as diagrams in which the (final) states are represented by labelled (double) circles and the transitions are represented by labelled arrows.

Example 2.8

Consider the DFA $\mathcal{M} = (\{q_0, q_1, q_{rej}\}, \{1,2\}, \delta, q_0, \{q_0, q_1\})$ where the transition function δ is represented by the following automata diagram:



The language recognised by $\mathcal{M}$ is the set of strings constructed by a sequence of 1s that preceeds a sequence of 2s. For instance, the empty string, 1, 12, 112, 122, and so on, are in $L_{\mathcal{M}}$.

We define the size of the DFA $\mathcal{M}$, denoted by $|\mathcal{M}|$, as the number of transitions, i.e., the product of $|Q|$ and $|\Sigma|$.

2.3 Constraint Satisfaction Problems

Many tasks from the declarative problem solving domain can be defined as constraint satisfaction problem (CSP; Dechter, 2003; Rossi et al., 2006), that are, combinatorial problems defined as a set of variables whose value must satisfy a number of limitations (the constraints).

Formally, a CSP is a triple $(\mathsf{V}, \mathsf{D}, \mathsf{C})$ where $\mathsf{V} = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ is a set of *(constraint) variables*, $\mathsf{D} = \{\mathrm{dom}(\mathsf{v}_1), \ldots, \mathrm{dom}(\mathsf{v}_n)\}$ is a set of the respective finite domains of values, and C is a finite set of constraints.

Variables

Intuitively, the domain of a variable represents all possible values that can be assumed from this variable. We assume that a variable has an ordered, discrete domain $\mathrm{dom}(\mathsf{v}) = [lb(\mathsf{v}), ub(\mathsf{v})]$ where $lb(\mathsf{v})$ denotes the smallest value and $ub(\mathsf{v})$ the greatest value. In particular, we will consider values from an integer interval. If for a variable v the associated domain is a singleton we say that the value of v is *fixed*.

We also consider *set variables* and *graph variables* (Dooms et al., 2005). A set variable represents a set of elements, whilst a graph variable represents a directed

graph. A set variable v has a discrete domain $\mathrm{dom}(\mathsf{v}) = [lb(\mathsf{v}), ub(\mathsf{v})]$ constructed from two sets, the *mandatory set* $lb(\mathsf{v})$ and the *possible set* $ub(\mathsf{v}) \setminus lb(\mathsf{v})$ of elements. In particular, v can take as value a set S such that $lb(\mathsf{v}) \subseteq S \subseteq ub(\mathsf{v})$. Similarly, the domain of a graph variable is given via graph inclusion. Thus, a graph variable v has a domain $\mathrm{dom}(\mathsf{v}) = [lb(\mathsf{v}), ub(\mathsf{v})]$ constructed from the lattice of graphs included between the *lower bound graph* $lb(\mathsf{v})$ and the *upper bound graph* $ub(\mathsf{v})$. A graph variable v can take as value a graph G such that G is a super graph of $lb(\mathsf{v})$ and a sub-graph of $ub(\mathsf{v})$.

Constraints

A *constraint* c is a k-ary relation, denoted $\mathrm{range}(c)$, on the domains of the variables in $\mathrm{scope}(c) \in \mathsf{V}^k$. The number of variables in $\mathrm{scope}(c)$ determines the *arity* of the constraint. Hence, a *binary* constraint has $|\mathrm{scope}(c)| = 2$, whilst an n-ary constraint has parametrised scope and is therefore called a *global* constraint. A *(constraint variable) assignment* is a function A that assigns to each variable $\mathsf{v} \in \mathsf{V}$ a value from $\mathrm{dom}(v)$. For a sequence of variables $\mathsf{v}_1, \ldots, \mathsf{v}_k$, define $A(\mathsf{v}_1, \ldots, \mathsf{v}_k) = (A(\mathsf{v}_1), \ldots, A(\mathsf{v}_k))$ and $A(\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle) = A(\mathsf{v}_1) \ldots A(\mathsf{v}_k)$. A constraint c is *satisfied* w.r.t. an assignment A if $A(\mathrm{scope}(c)) \in \mathrm{range}(c)$. Otherwise, we say that c is *violated*. The *complement* of a constraint c is denoted by $\overline{c}$. It is easy to see that for any assignment, c is violated if and only if $\overline{c}$ is satisfied. We denote the subset of constraints from C satisfied w.r.t. A by

$$\mathrm{sat}_{\mathsf{C}}(A) = \{c \mid c \in \mathsf{C},\ A(\mathrm{scope}(c)) \in \mathrm{range}(c)\}.$$

An assignment A is a *solution* for the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$ if $\mathsf{C} = \mathrm{sat}_{\mathsf{C}}(A)$, i.e., all constraints are satisfied.

In what follows, we are more interested at pairing an assignment with a configuration of satisfied and violated constraints. A *configuration* is a pair $(A, \mathrm{sat}_{\mathsf{C}}(A))$ where A is an assignment and $\mathrm{sat}_{\mathsf{C}}(A)$ is the set of constraints satisfied w.r.t. A. Obviously, the set of constraints violated w.r.t. A is $\mathsf{C} \setminus \mathrm{sat}_{\mathsf{C}}(A)$. It is easy to see that the problem of finding configurations for a CSP can be reformulated to the task of finding solutions for a CSP.

Local Consistency

Constraint programming (CP) systems are oriented towards solving CSP and typically interleave backtracking search to explore assignments in a *search tree* with

constraint propagation to prune the set of values a variable can take (Dechter, 2003; Rossi et al., 2006). In a search tree, each vertex represents a partial assignment to only some variables. Child vertices are typically obtained by selecting an unassigned variable and having two child vertices, one for a possible value assigned to this variable, and one with the domain of the variable reduced by the value. Other types of branching are also possible, e.g., *domain splitting* (Dincbas et al., 1988). Every time a domain changes, a constraint propagation stage is executed, pruning the set of values for the other variables. The effect of constraint propagation is studied in terms of *local consistency*.

A binary constraint c is *arc consistent* if whenever a variable v in $\text{scope}(c)$ is assigned any value, there exists a value in the domain for the other variable in $\text{scope}(c) \setminus \{\mathsf{v}\}$ such that c is not violated. An n-ary constraint c is *domain consistent* if whenever a variable $\mathsf{v}_i \in \text{scope}(c) = \{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$ is assigned any value $d_i \in \text{dom}(\mathsf{v}_i)$, there exist compatible values in the domains of all the other variables, i.e., $d_j \in \text{dom}(\mathsf{v}_j)$, where $1 \leq j \leq n,\ j \neq i$, such that $(d_1, \ldots, d_n) \in \text{range}(c)$, forming an *($n$-ary) support* for $\mathsf{v}_i = d_i$. Any n-ary constraint can be *decomposed* into a k-ary constraint with $k \geq 2$ (Montanari, 1974). Often, binary decompositions ($k = 2$) are considered. Observe that in general, however, a constraint propagator that enforces domain consistency prunes more values than one that enforces arc consistency on a binary decomposition of the original constraint (Walsh, 2000).

Bound and range consistency are defined for constraints over finite intervals. A constraint c is *bound consistent* if whenever a variable $\mathsf{v} \in \text{scope}(c)$ is assigned the smallest value $lb(\mathsf{v})$ or the greatest value $ub(\mathsf{v})$ of its domain, there exist consistent values between the smallest and greatest value for all the other variables in the scope of the constraint, called a *bound support*. A constraint is *range consistent* if whenever a variable is assigned any value in its domain, there exists a bound support. Range consistency is in between domain and bound consistency, where domain consistency is the strongest of the three local consistency properties.

2.4 Boolean Satisfiability

A special case of CSP is Boolean satisfiability (SAT; Biere et al., 2009), i.e., where all variables are propositional and, in our context, all constraints are forbidden combinations of literals called nogoods.

Using the notation of Gebser and Schaub (2013), a *literal* is a formula of the form $\mathbf{T}p$ or $\mathbf{F}p$, where p is a propositional variable. Intuitively, $\mathbf{T}p$ expresses that p

is assigned *true* and $\mathbf{F}p$ that it is *false*. The *complement* of a literal σ is denoted $\overline{\sigma}$, i.e., $\overline{\mathbf{T}p} = \mathbf{F}p$ and $\overline{\mathbf{F}p} = \mathbf{T}p$. To access the variable in a literal σ we write $\mathrm{var}(\sigma)$, e.g., $\mathrm{var}(\mathbf{T}p) = p$.

A *(Boolean) assignment* $\mathbf{A}$ over a set of propositional variables $\mathrm{dom}(\mathbf{A})$, is a set of literals. To access the true and the false variables in $\mathbf{A}$ we use $\mathbf{A}^{\mathbf{T}} = \{p \mid \mathbf{T}p \in \mathbf{A}\}$ and $\mathbf{A}^{\mathbf{F}} = \{p \mid \mathbf{F}p \in \mathbf{A}\}$. We say that $\mathbf{A}$ is *conflict-free* if $\mathbf{A}^{\mathbf{T}} \cap \mathbf{A}^{\mathbf{F}} = \emptyset$, otherwise $\mathbf{A}$ is *conflicting*. For two assignments $\mathbf{A}$ and $\mathbf{A}'$, $\mathbf{A}$ *extends* $\mathbf{A}'$ if $\mathbf{A} \supseteq \mathbf{A}'$. Furthermore, $\mathbf{A}$ is *total* if $\mathbf{A}^{\mathbf{T}} \cup \mathbf{A}^{\mathbf{F}} = \mathrm{dom}(\mathbf{A})$, otherwise $\mathbf{A}$ is partial.

A *nogood* is a set $\{\sigma_1, \ldots, \sigma_k\}$ of literals σ_i for $1 \leq i \leq k$, expressing a forbidden combination of literals. Accordingly, for an assignment $\mathbf{A}$, a nogood δ is *conflicting* if $\delta \subseteq \mathbf{A}$. For a set of nogoods Δ, a total and conflict-free assignment $\mathbf{A}$ is a *solution* if $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta$.

Example 2.9

Consider the set of nogoods $\Delta = \{\{\mathbf{T}p, \mathbf{T}q\}, \{\mathbf{F}p, \mathbf{F}q\}\}$ and the assignments $\mathbf{A} = \{\mathbf{T}p\}$, $\mathbf{A}' = \{\mathbf{T}p, \mathbf{F}q\}$, and $\mathbf{A}'' = \{\mathbf{T}p, \mathbf{T}q\}$.

Observe that $\mathbf{A}'$ extends $\mathbf{A}$. Since $\mathbf{A}'$ is total and conflict-free, we have that $\mathbf{A}'$ is a solution for Δ. On the other hand, the assignment $\mathbf{A}''$ also extends $\mathbf{A}$, but the nogood $\{\mathbf{T}p, \mathbf{T}q\}$ is conflicting for $\mathbf{A}''$, since $\{\mathbf{T}p, \mathbf{T}q\} \subseteq \mathbf{A}''$.

Unit Propagation

Unit propagation (UP) is the most important inference mechanism in SAT (Biere et al., 2009). It forces the inclusion of a literal into an (extended) assignment if its complement occurs in a nogood whose other literals are already included.

Formally, given an assignment $\mathbf{A}$, for a nogood δ and a literal $\sigma \in \delta$, if $\delta \setminus \{\sigma\} \subseteq \mathbf{A}$ and $\overline{\sigma} \notin \mathbf{A}$ then δ is *unit* w.r.t. $\mathbf{A}$ and $\overline{\sigma}$ is *unit-resulting*, i.e., only unit-resulting literals can avert $\delta \subseteq \mathbf{A}$. UP is the process of extending an assignment with unit-resulting literals.

Definition 2.1: Unit Propagation

Given an assignment $\mathbf{A}$ and a set of nogoods Δ, we define

$$\mathrm{UP}(\Delta, \mathbf{A}) = \begin{cases} \mathbf{A} \cup \{\sigma\} & \text{if } \sigma \text{ is unit-resulting w.r.t. } \mathbf{A} \text{ for some } \delta \in \Delta, \\ \mathbf{A} & \text{otherwise.} \end{cases}$$

Note that, in general, there might be several choices for σ. Therefore, we will consider the fixpoint of UP by default. Only in formulas will we denote it by $\mathrm{UP}^*(\Delta, \mathbf{A})$.

Example 2.10

Reconsider the set of nogoods $\Delta = \{\{\mathbf{T}p, \mathbf{T}q\}, \{\mathbf{F}p, \mathbf{F}q\}\}$ and the assignment $\mathbf{A} = \{\mathbf{T}p\}$ from Example 2.9. We have that the nogood $\{\mathbf{T}p, \mathbf{T}q\}$ is unit w.r.t. $\mathbf{A}$, and $\mathbf{F}q$ is unit-resulting. Hence, $\mathrm{UP}(\Delta, \mathbf{A}) = \mathbf{A}' = \{\mathbf{T}p, \mathbf{F}q\}$. Note that this is also the fixpoint of UP as $\mathrm{UP}(\Delta, \mathbf{A}') = \mathbf{A}'$.

There exist very efficient implementations of UP, for instance, the *watched literals* method (Moskewicz et al., 2001). Down any branch of the search tree, the propagation time is proportional to the number of literals in the nogoods.

Theorem 2.1: Complexity of Unit Propagation (Dechter, 2003)

UP takes $\mathcal{O}(k)$ time down any branch of the search tree to propagate a set of nogoods with a total size of k literals.

Under certain conditions, the fixpoint application of UP achieves a solution for Δ. Given an assignment $\mathbf{A}$, we say that a nogood δ is *Horn-style* if $\delta \setminus \mathbf{A}$ is of the form $\{\mathbf{T}p_0, \mathbf{F}p_1, \ldots, \mathbf{F}p_m\}$, i.e., δ contains at most one positive literal of an unassigned variable. If all nogoods in Δ are Horn-style then UP finds a solution of Δ in linear time (cf. Dowling and Gallier, 1984).

Conflict-driven Nogood Learning

An efficient decision-algorithm for solving SAT is conflict-driven nogood learning (CDNL; Bayardo and Schrag, 1997; Marques-Silva and Sakallah, 1999). It combines search and UP, and is centred around ConflictAnalysis (Moskewicz et al., 2001), a process of extracting a reason for a conflicting assignment encountered during search by iteratively resolving a conflicting nogood against other nogoods. This guides *backjumping*, a form of non-chronological backtracking that may skip some *decision level*. Recording the extracted reason in a nogood facilitates *conflict-driven learning*. This can prune the search space and lead to more propagation.

We define CDNL as shown in Algorithm 2.1 in accordance with Gebser et al. (2007a). Given a set of nogoods Δ, CDNL starts from an empty assignment $\mathbf{A}$, and recorded nogoods ∇, at decision level dl initialised with 0.

First, CDNL applies UP to extend $\mathbf{A}$ (Line 5). If this encounters a conflict in Line 6 then ConflictAnalysis generates a nogood ε and determines a decision level k at which to continue search (Line 8). Then, ε is added to the set of recorded nogoods ∇ in Line 9. Lines 10–11 account for backjumping to level k, where $dl(\sigma)$

Input : A set of nogoods Δ.
Output: A solution for Δ if one exists.

```
1  A ← ∅                                                  // assignment
2  ∇ ← ∅                                                  // recorded nogoods
3  dl ← 0                                                 // decision level
4  loop
5      A ← UP*(Δ ∪ ∇, A)
6      if δ ⊆ A for some δ ∈ Δ then
7          if dl = 0 then return no solution
8          (ε, k) ← ConflictAnalysis(δ, Δ ∪ ∇, A)
9          ∇ ← ∇ ∪ {ε}
10         A ← A \ {σ ∈ A | k < dl(σ)}
11         dl ← k
12     else if A^T ∪ A^F = dom(A) then
13         return A
14     else
15         σ_d ← Select(Δ, ∇, A)
16         A ← A ∪ {σ_d}
17         dl ← dl + 1
```

Algorithm 2.1: CDNL

is used to access the decision level of a literal σ. If ConflictAnalysis, however, yields a conflict at level 0, no solution exists (Line 7).

Second, if $\mathbf{A}$ is total then it is a solution of Δ (Lines 12–13). Otherwise (Lines 14–17), $\mathbf{A}$ is partial and no nogood is conflicting. Then, a decision literal σ_d is selected by some heuristic, denoted by Select, i.e., based on the information from Δ, recorded nogoods ∇, or the current assignment $\mathbf{A}$. The literal is added to $\mathbf{A}$, and the decision level is incremented.

2.5 Answer Set Programming

Another special case of CSP is answer set programming (ASP; Baral, 2003; Gebser et al., 2012b), i.e., where all variables are propositional and the conditions on them are induced by a logic program under answer set semantics. In this thesis, we consider normal logic programs. Given a finite set of atomic propositions $\mathscr{P}$, a *(propositional normal logic) program* Π is a finite set of *rules* r of the form

$$p_0 \leftarrow p_1, \ldots, p_m, \; not \; p_{m+1}, \ldots, \; not \; p_n \tag{2.1}$$

where each $p_i \in \mathscr{P}$ is an *atom* for $0 \leq i \leq n$. A *default literal* is an atom p or its *default negation* $not\ p$. Furthermore, for a rule r of the form (2.1), we define

$$\text{head}(r) = p_0$$
$$\text{body}(r) = \{p_1, \ldots, p_m,\ not\ p_{m+1}, \ldots,\ not\ p_n\}$$

where the atom head(r) is called the *head* of r, and the set of default literals body(r) is called the *body* of r. For any set $S = \{p_1, \ldots, p_m,\ not\ p_{m+1}, \ldots,\ not\ p_n\}$, define $S^+ = \{p_1, \ldots, p_m\}$ and $S^- = \{p_{m+1}, \ldots, p_n\}$. Given this, for any rule r, the sets of atoms body$(r)^+$ is called the *positive body* of r and body$(r)^-$ is called the *negative body* of r. If body$(r) = \emptyset$ then r is called a *fact*. For a program Π we define the following notations:

$$\text{atom}(\Pi) = \bigcup_{r \in \Pi} (\{\text{head}(r)\} \cup \text{body}(r)^+ \cup \text{body}(r)^-$$
$$\text{body}(\Pi) = \{\text{body}(r) \mid r \in \Pi\}$$

The sets atom(Π) and body(Π) denote the atoms and bodies that occur in Π, respectively. To access rules sharing heads from a set X, define

$$\Pi_X = \{r \in \Pi \mid \text{head}(r) \in X\}.$$

Note that Π_X itself is a program.

The semantics of a program Π is given by its answer sets, which have been defined in many interchangeable ways (Lifschitz, 2008b). Traditionally, the answer sets are viewed as the entailed atoms in classical models that are necessarily supported by an applicable rule. For a set of atoms X, a rule r is *applicable* w.r.t. X if body$(r)^+ \subseteq X$ and body$(r)^- \cap X = \emptyset$. If r is applicable then head(r) is *supported* by r w.r.t. X. If every atom $p \in X$ is supported by some rule in Π w.r.t. X, we call X *supported* by Π. For our purposes, we say that a set of atoms X is a *model* of Π if either head$(r) \in X$, body$(r)^+ \not\subseteq X$, or body$(r)^- \cap X \neq \emptyset$ holds for every rule $r \in \Pi$. Furthermore, a model X of Π that is supported by Π is called a *supported model* of Π.

Definition 2.2: Answer Set of a Program

Given a program Π, a set $X \subseteq$ atom(Π) is an *answer set* of Π, if X is the least model of the *reduct* (Gelfond and Lifschitz, 1988):

$$\Pi^X = \{\text{head}(r) \leftarrow \text{body}(r)^+ \mid r \in \Pi,\ \text{body}(r)^- \cap X = \emptyset\}.$$

Note that, since the negative body of every rule in Π^X is empty, the least model of Π^X is unique. Moreover, every answer set of Π is a supported model of Π. The converse, however, does not hold in general (Fages, 1994). The following example will serve as a running example.

Example 2.11

Consider the program Π, given through the set of rules

$$r_1 : p \leftarrow not\ q \qquad r_3 : r \leftarrow p \qquad r_5 : s \leftarrow r$$
$$r_2 : q \leftarrow not\ p \qquad r_4 : r \leftarrow s$$

and the sets $X_1 = \{p, r, s\}$, $X_2 = \{q, r, s\}$, and $X_3 = \{q\}$, all of which are supported models of Π. We obtain their corresponding reducts by removing all rules from Π whose negative body contains an atom from a set, and then dropping all default negated atoms from the remaining rules. Hence, we have

$$\Pi^{X_1} = \left\{ \begin{array}{l} p \leftarrow \\ r \leftarrow p \\ r \leftarrow s \\ s \leftarrow r \end{array} \right\}, \qquad \Pi^{X_2} = \left\{ \begin{array}{l} q \leftarrow \\ r \leftarrow s \\ s \leftarrow r \end{array} \right\}, \qquad \Pi^{X_3} = \left\{ \ q \leftarrow \ \right\}.$$

Verify that X_1 is the least model of Π^{X_1} and X_3 is the least model of Π^{X_3}. On the other hand, the set $\{q\} \subset X_2$ is the least model of Π^{X_2}. Hence, X_1 and X_3 are answer sets of Π, whilst X_2 is not.

We often use an alternative characterisation of answer sets. In particular, the one of Lee (2005), who has shown that the answer sets of a program Π coincide with the supported models Π that do not contain a non-empty unfounded set.

In accordance with Gebser et al. (2007a), we view a model X of Π as an assignment $\mathbf{A}$ with $\mathrm{dom}(\mathbf{A}) = \mathrm{atom}(\Pi) \cup \mathrm{body}(\Pi)$, in particular, $p \in \mathbf{A^T}$ whenever $p \in X$ and $p \in \mathbf{A^F}$ whenever $p \notin X$ where $p \in \mathrm{atom}(\Pi)$, and represent the conditions induced by the completion of Π (Clark, 1978) in terms of nogoods. These capture the supported models of Π (cf. Apt et al., 1988) and allow for their computation by means of CDNL.

To begin, for a set of default literals $\beta = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, define

$$\mathrm{EQ}_\beta = \left\{ \begin{array}{l} \{\mathbf{F}\beta, \mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}, \\ \{\mathbf{F}p_1, \mathbf{T}\beta\}, \ldots, \{\mathbf{F}p_m, \mathbf{T}\beta\}, \{\mathbf{T}p_{m+1}, \mathbf{T}\beta\}, \ldots, \{\mathbf{T}p_n, \mathbf{T}\beta\} \end{array} \right\}.$$

Typically, we have $\beta \in \text{body}(\Pi)$. Then, the nogoods in EQ_β represent that β must be false if and only if some of its literals does not hold. For an assignment $\mathbf{A}$, if at least the implication holds for every rule in Π, i.e., if $\{\beta \in \text{body}(\Pi) \mid (\beta^+ \cap \mathbf{A}^\mathbf{F}) \cup (\beta^- \cap \mathbf{A}^\mathbf{T}) \neq \emptyset\} \subseteq \mathbf{A}^\mathbf{F}$, then we say that $\mathbf{A}$ is *body-saturated*. Observe that running UP on the nogoods $\bigcup_{\beta \in \text{body}(\Pi)} \text{EQ}_\beta$ to a fixpoint achieves a body-saturated assignment.

For an atom $p \in \text{atom}(\Pi)$, let $\{\beta_1, \ldots, \beta_k\} = \text{body}(\Pi_{\{p\}})$, i.e., the bodies of rules with head p. We define

$$\mathbf{\Delta}^p_\Pi = \left\{ \begin{array}{l} \{\mathbf{T}\beta_1, \mathbf{F}p\}, \ldots, \{\mathbf{T}\beta_k, \mathbf{F}p\}, \\ \lambda^p_{\text{body}(\Pi_{\{p\}})} \end{array} \right\}$$

where for any set of bodies $S = \{\beta_1, \ldots, \beta_k\}$

$$\lambda^p_S = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}.$$

The nogoods in $\{\mathbf{T}\beta_1, \mathbf{F}p\}, \ldots, \{\mathbf{T}\beta_k, \mathbf{F}p\}$ view the rules in Π as implications. On the other hand, the nogood $\lambda^p_{\text{body}(\Pi_{\{p\}})}$ stipulates the existence of a rule in Π that supports p if p is true. To put it in different words, $\lambda^p_{\text{body}(\Pi_{\{p\}})}$ represents that p cannot hold if p is not supported by any rule in Π.

Definition 2.3: Completion Nogoods

For a program Π, the *completion nogoods* are

$$\Delta_\Pi = \bigcup_{\beta \in \text{body}(\Pi)} \text{EQ}_\beta \cup \bigcup_{p \in \text{atom}(\Pi)} \mathbf{\Delta}^p_\Pi.$$

As Gebser et al. (2007a) have shown, the solutions for Δ_Π correspond to the supported models of Π.

Theorem 2.2: Completion Nogoods of a Program (Gebser et al., 2007a)

Let Π be a program and $X \subseteq \text{atom}(\Pi)$ and $\mathbf{A}$ the assignment given through

$$\begin{aligned} \mathbf{A} = &\{\mathbf{T}p \mid p \in X\} \cup \{\mathbf{F}p \mid p \in \text{atom}(\Pi) \setminus X\} \\ &\cup \{\mathbf{T}\text{body}(r) \mid r \in \Pi, \text{body}(r)^+ \subseteq X, \text{body}(r)^- \cap X = \emptyset\} \\ &\cup \{\mathbf{F}\text{body}(r) \mid r \in \Pi, (\text{body}(r)^+ \cap (\text{atom}(\Pi) \setminus X)) \\ &\qquad \cup (\text{body}(r)^- \cap X) \neq \emptyset\}. \end{aligned}$$

Then, X is a supported model of Π if and only if $\mathbf{A}$ is a solution for Δ_Π.

A direct consequence from the theorem is that we can apply CDNL to compute models of Π's completion. We illustrate the theorem with an example.

Example 2.12

Reconsider the program Π from Example 2.11. For the bodies in body(Π) we have the following equations:

$$\mathrm{EQ}_{\mathrm{body}(r_1)}\,\Pi = \{\{\mathbf{F}\,\mathrm{body}(r_1), \mathbf{F}q\}, \{\mathbf{T}\,\mathrm{body}(r_1), \mathbf{T}q\}\}$$
$$\mathrm{EQ}_{\mathrm{body}(r_2)}\,\Pi = \{\{\mathbf{F}\,\mathrm{body}(r_2), \mathbf{F}p\}, \{\mathbf{T}\,\mathrm{body}(r_2), \mathbf{T}p\}\}$$
$$\mathrm{EQ}_{\mathrm{body}(r_3)}\,\Pi = \{\{\mathbf{F}\,\mathrm{body}(r_3), \mathbf{T}p\}, \{\mathbf{T}\,\mathrm{body}(r_3), \mathbf{F}p\}\}$$
$$\mathrm{EQ}_{\mathrm{body}(r_4)}\,\Pi = \{\mathbf{F}\,\mathrm{body}(r_4), \mathbf{T}s\}, \{\mathbf{T}\,\mathrm{body}(r_4), \mathbf{F}s\}\}$$
$$\mathrm{EQ}_{\mathrm{body}(r_5)}\,\Pi = \{\{\{\mathbf{F}\,\mathrm{body}(r_5), \mathbf{T}r\}, \{\mathbf{T}\,\mathrm{body}(r_5), \mathbf{F}r\}\}$$

For the atoms in atom(Π) we have the following equations:

$$\mathbf{\Delta}^p_\Pi = \{\{\mathbf{T}p, \mathbf{F}\,\mathrm{body}(r_1)\}, \{\mathbf{F}p, \mathbf{T}\,\mathrm{body}(r_1)\}\}$$
$$\mathbf{\Delta}^q_\Pi = \{\{\mathbf{T}q, \mathbf{F}\,\mathrm{body}(r_2)\}, \{\mathbf{F}q, \mathbf{T}\,\mathrm{body}(r_2)\}\}$$
$$\mathbf{\Delta}^r_\Pi = \{\{\mathbf{T}r, \mathbf{F}\,\mathrm{body}(r_3), \mathbf{F}\,\mathrm{body}(r_4)\}, \{\mathbf{F}r, \mathbf{T}\,\mathrm{body}(r_3)\}, \{\mathbf{F}r, \mathbf{T}\,\mathrm{body}(r_4)\}\}$$
$$\mathbf{\Delta}^s_\Pi = \{\{\mathbf{T}s, \mathbf{F}\,\mathrm{body}(r_5)\}, \{\mathbf{F}s, \mathbf{T}\,\mathrm{body}(r_5)\}\}$$

Altogether, the completion nogoods of Π are given through

$$\Delta_\Pi = \mathrm{EQ}_{\mathrm{body}(r_1)}\,\Pi \cup \cdots \cup \mathrm{EQ}_{\mathrm{body}(r_5)}\,\Pi \cup \mathbf{\Delta}^p_\Pi \cup \mathbf{\Delta}^q_\Pi \cup \mathbf{\Delta}^r_\Pi \cup \mathbf{\Delta}^s_\Pi.$$

Now, recall that the sets $X_1 = \{p, r, s\}$, $X_2 = \{q, r, s\}$, and $X_3 = \{q\}$ are supported models of Π. In accordance with Theorem 2.2, we represent X_1, X_2, and X_3 by the respective assignments $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ as follows:

$$\mathbf{A}_1 = \{\mathbf{T}p, \mathbf{F}q, \mathbf{T}r, \mathbf{T}s, \mathbf{T}\,\mathrm{body}(r_1), \mathbf{F}\,\mathrm{body}(r_2), \mathbf{T}\,\mathrm{body}(r_3), \mathbf{T}\,\mathrm{body}(r_4), \mathbf{T}\,\mathrm{body}(r_5)\}$$
$$\mathbf{A}_2 = \{\mathbf{F}p, \mathbf{T}q, \mathbf{T}r, \mathbf{T}s, \mathbf{F}\,\mathrm{body}(r_1), \mathbf{T}\,\mathrm{body}(r_2), \mathbf{F}\,\mathrm{body}(r_3), \mathbf{T}\,\mathrm{body}(r_4), \mathbf{T}\,\mathrm{body}(r_5)\}$$
$$\mathbf{A}_3 = \{\mathbf{F}p, \mathbf{T}q, \mathbf{F}r, \mathbf{F}s, \mathbf{F}\,\mathrm{body}(r_1), \mathbf{T}\,\mathrm{body}(r_2), \mathbf{F}\,\mathrm{body}(r_3), \mathbf{F}\,\mathrm{body}(r_4), \mathbf{F}\,\mathrm{body}(r_5)\}$$

Verify that $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ are solutions for Δ_Π, i.e., they are total and no nogood is violated.

Since the nogoods in Δ_Π are solely determined by Π, we sometimes abuse notation and write that we apply UP on Π, rather than UP on Δ_Π.

We now turn to unfounded sets (Van Gelder et al., 1991). They characterise atoms in a program that might circularly support themselves when they have no supporting rule that is external to the set. Formally, for a program Π and a set

$U \subseteq \text{atom}(\Pi)$, the *external support* of U is defined as

$$\text{ES}_\Pi(U) = \{\text{body}(r) \mid r \in \Pi,\ \text{head}(r) \in U,\ \text{body}(r)^+ \cap U = \emptyset\}.$$

Definition 2.4: Unfounded Set

Let Π be a program and $\mathbf{A}$ an assignment. A set $U \subseteq \text{atom}(\Pi)$ is an *unfounded set* of Π w.r.t. $\mathbf{A}$ if $\text{ES}_\Pi(U) \subseteq \mathbf{A}^\mathbf{F}$.

Furthermore, we say that $\mathbf{A}$ is *unfounded-free* if for all $U \subseteq \text{atom}(\Pi)$ such that $\text{ES}_\Pi(U) \subseteq \mathbf{A}^\mathbf{F}$ we have that $U \subseteq \mathbf{A}^\mathbf{F}$, i.e., all atoms from unfounded sets are false.

We now have all building blocks at hand, to formulate a first alternative characterisation of a program's answer sets. Following Lee (2005), the answer sets of a program Π coincide with the supported models of Π that do not contain a non-empty unfounded set. In light of Theorem 2.2, we obtain the following variant.

Theorem 2.3

Let Π be a program and $X \subseteq \text{atom}(\Pi)$ and $\mathbf{A}$ the assignment given through

$$\begin{aligned}\mathbf{A} = &\{\mathbf{T}p \mid p \in X\} \cup \{\mathbf{F}p \mid p \in \text{atom}(\Pi) \setminus X\}\\ &\cup\{\mathbf{T}\,\text{body}(r) \mid r \in \Pi, \text{body}(r)^+ \subseteq X, \text{body}(r)^- \cap X = \emptyset\}\\ &\cup\{\mathbf{F}\,\text{body}(r) \mid r \in \Pi, (\text{body}(r)^+ \cap (\text{atom}(\Pi) \setminus X))\\ &\qquad\qquad \cup (\text{body}(r)^- \cap X) \neq \emptyset\}.\end{aligned}$$

Then, X is an answer set of Π if and only if $\mathbf{A}$ is a solution for Δ_Π and unfounded-free for Π.

We illustrate the theorem in the following example.

Example 2.13

We proceed from Example 2.12. Verify that $\mathbf{A}_1$ and $\mathbf{A}_3$ are unfounded-free for Π. On the other hand, for $U = \{r, s\}$ we have $\text{ES}_\Pi(U) = \{\text{body}(r_3)\} \subseteq \mathbf{A}_2^\mathbf{F}$. That is, U is an unfounded set for Π w.r.t. $\mathbf{A}_2$.

Therefore, by Theorem 2.3, only sets X_1 and X_3 are answer sets of Π. Indeed, they are (cf. Example 2.11).

Attention is often restricted to unfounded sets that are subsets of SCCs (i.e., loops) in the *(positive) dependency graph* of Π, denoted $\text{DG}^+(\Pi)$, where $\text{DG}^+(\Pi) = (V, E)$ is

defined by

$$V = \text{atom}(\Pi) \cup \text{body}(\Pi)$$
$$E = \{(\text{body}(r), \text{head}(r)) \mid r \in \Pi\} \cup \{(p, \text{body}(r)) \mid r \in \Pi,\ p \in \text{body}(r)^+\}).$$

A non-empty set of atoms $U \subseteq \text{atom}(\Pi)$ is a *loop* of Π if U is the set of vertices in a loop of $\text{DG}^+(\Pi)$ (Lee, 2005). We denote by $\text{loop}(\Pi)$ the set of all loops in Π, and define for $\beta \in \text{body}(\Pi)$ the set $\text{scc}(\beta)$ as being composed of all atoms that belong to the same SCC as β.

Example 2.14

Reconsider the program Π from Example 2.11. We have

$$\text{loop}(\Pi) = \{\{p\}, \{q\}, \{r\}, \{s\}, \{r, s\}\}.$$

If every loop of Π is a singleton, then we say that Π is *tight*. If Π is *tight*, then the supported models of Π are precisely the answer sets of Π (Erdem and Lifschitz, 2003).

Well-founded Negation and Well-founded Justification

An important inference operation that aims at unfounded sets is well-founded negation (WFN; Gebser and Schaub, 2013). WFN is the process of extending an assignment by assigning false to all atoms that are included in an unfounded set.

Definition 2.5: Well-founded Negation

For sets of atoms $\Omega \subseteq 2^{\text{atom}(\Pi)}$ we define

$$\text{WFN}[\Omega](\Pi, \mathbf{A}) = \begin{cases} \mathbf{A} \cup \{\mathbf{F}p\} & \text{if } U \in \Omega,\ p \in U,\ \text{ES}_\Pi(U) \subseteq \mathbf{A}^\mathbf{F}, \\ \mathbf{A} & \text{otherwise.} \end{cases}$$

By construction, if $\Omega = 2^{\text{atom}(\Pi)}$ then fixpoint operation of $\text{WFN}[\Omega](\Pi, \mathbf{A})$ achieves an unfounded-free assignment. In practice, it is enough to consider only unfounded sets that are loops, i.e., $\Omega = \text{loop}(\Pi)$, resulting in a restricted form of WFN referred to as forward loop (FL). Fixpoint operation of FL and UP, however, simulates the effect of WFN and UP (Gebser and Schaub, 2013). FL can be implemented such that it takes $\mathcal{O}(|\Pi|)$ time (cf. Anger et al., 2006). Formally, we denote the fixpoint of $\text{WFN}[\Omega](\Pi, \mathbf{A})$ by $\text{WFN}^*[\Omega](\Pi, \mathbf{A})$.

Example 2.15

Proceeding from Example 2.13. Recall that for $U = \{r, s\}$ we have $\mathrm{ES}_\Pi(U) = \{\mathrm{body}(r_3)\} \subseteq \mathbf{A}_2^{\mathbf{F}}$. Hence, WFN extends the assignment as follows:

$$\mathrm{WFN}^*[2^{\mathrm{atom}(\Pi)}](\Pi, \mathbf{A}_2) = \mathbf{A}_2 \cup \{\mathbf{F}r, \mathbf{F}s\}.$$

Note that the extended assignment is now conflicting, e.g., we have $\mathbf{T}r$ and $\mathbf{F}r$ in $\mathbf{A}_2 \cup \{\mathbf{F}r, \mathbf{F}s\}$. Since $U \in \mathrm{loop}(\Pi)$ (cf. Example 2.14), FL achieves the same result, i.e.,

$$\mathrm{WFN}^*[\mathrm{loop}(\Pi)](\Pi, \mathbf{A}_2) = \mathbf{A}_2 \cup \{\mathbf{F}r, \mathbf{F}s\}.$$

Most modern ASP solvers apply FL also on partial assignments. For instance, consider the assignment $\mathbf{A}_2' \subset \mathbf{A}_2$ given through $\mathbf{A}_2' = \{\mathbf{F}p, \mathbf{F}\mathrm{body}(r_3)\}$. Verify that for $U = \{r, s\}$ we have $\mathrm{ES}_\Pi(U) = \{\mathrm{body}(r_3)\} \subseteq (\mathbf{A}')_2^{\mathbf{F}}$, and FL extends the assignment in a similar way:

$$\mathrm{WFN}^*[\mathrm{loop}(\Pi)](\Pi, \mathbf{A}_2') = \mathbf{A}_2' \cup \{\mathbf{F}r, \mathbf{F}s\}.$$

The contrapositive of WFN is well-founded justification (WFJ; Gebser and Schaub, 2013). It establishes the truth of the only remaining external support of a set of atoms which contains at least one true atom.

Definition 2.6: Well-founded Justification

For sets of atoms $\Omega \subseteq 2^{\mathrm{atom}(\Pi)}$ we define

$$\mathrm{WFJ}[\Omega](\Pi, \mathbf{A}) = \begin{cases} \mathbf{A} \cup \{\mathbf{T}\beta\} & \text{if } U \in \Omega,\ p \in U \cap \mathbf{A}^{\mathbf{T}},\ \mathrm{ES}_\Pi(U) \setminus \mathbf{A}^{\mathbf{F}} = \{\beta\}, \\ \mathbf{A} & \text{otherwise.} \end{cases}$$

Again, we consider the alternatives $\Omega = 2^{\mathrm{atom}(\Pi)}$, i.e., what we refer to as WFJ, and $\Omega = \mathrm{loop}(\Pi)$, called backward loop (BL). Similar to UP and WFN, there might be several choices for extending an assignment with WFJ and BL. Therefore, we will consider their respective fixpoints.

Example 2.16

Proceeding from Example 2.13, for instance, consider the partial assignment $\mathbf{A}_1' \subset \mathbf{A}_1$ given through $\mathbf{A}_1' = \{\mathbf{T}r\}$. Verify that for $U = \{r, s\}$ we have $\mathrm{ES}_\Pi(U) = \{\mathrm{body}(r_3)\}$ and $\mathrm{body}(r_3) \notin \mathbf{A}_1^{\mathbf{F}}$. Hence, WFJ extends the assignment as follows:

$$\mathrm{WFJ}[2^{\mathrm{atom}(\Pi)}](\Pi, \mathbf{A}_1') = \mathbf{A}_1' \cup \{\mathbf{T}\mathrm{body}(r_3)\}.$$

Since $U \in \text{loop}(\Pi)$ (cf. Example 2.14), BL achieves the same result, i.e.,

$$\text{WFJ}[\text{loop}(\Pi)](\Pi, \mathbf{A}'_1) = \mathbf{A}'_1 \cup \{\mathbf{T}\,\text{body}(r_3)\}.$$

Gebser and Schaub have shown that, in general, WFJ propagates more consequences than BL. The time complexity for computing WFJ is $\mathcal{O}(|\Pi|^2)$, a relatively high computational cost, as it amounts to failed-literal-detection and WFN.

Reduction to Boolean Satisfiability

In principle, the unfounded set conditions can be represented by sets of nogoods. In accordance with Lee (2005), the set of loop nogoods of a program (Gebser et al., 2007a) express that the atoms in every unfounded set have to be falsified.

Definition 2.7: Loop Nogoods of a Program

Let Π be a program. The set of *loop nogoods* of Π, denoted by Λ_Π, is

$$\Lambda_\Pi = \bigcup_{U \subseteq \text{atom}(\Pi)} \{\lambda^p_{\text{ES}_\Pi(U)} \mid p \in U\}.$$

As Gebser et al. have shown, the answer sets of a program Π can be characterised by the solutions to the completion nogoods and the loop nogoods of Π.

Theorem 2.4: Reduction to Boolean Satisfiability (Gebser et al., 2007a)

Let Π be a program and $X \subseteq \text{atom}(\Pi)$ and $\mathbf{A}$ the assignment given through

$$\begin{aligned}\mathbf{A} = &\{\mathbf{T}p \mid p \in X\} \cup \{\mathbf{F}p \mid p \in \text{atom}(\Pi) \setminus X\}\\ &\cup \{\mathbf{T}\,\text{body}(r) \mid r \in \Pi, \text{body}(r)^+ \subseteq X, \text{body}(r)^- \cap X = \emptyset\}\\ &\cup \{\mathbf{F}\,\text{body}(r) \mid r \in \Pi, (\text{body}(r)^+ \cap (\text{atom}(\Pi) \setminus X))\\ &\qquad\qquad \cup (\text{body}(r)^- \cap X) \neq \emptyset\}.\end{aligned}$$

Then, X is an answer set of Π if and only if $\mathbf{A}$ is a solution for $\Delta_\Pi \cup \Lambda_\Pi$.

Example 2.17

Reconsider the program Π from Example 2.11. For the non-trivial loops from $\text{loop}(\Pi)$ we have the following equations:

$$\lambda^r_{\text{ES}_\Pi(\{r,s\})} = \{\mathbf{T}r, \mathbf{F}\,\text{body}(r_3)\}$$
$$\lambda^s_{\text{ES}_\Pi(\{r,s\})} = \{\mathbf{T}s, \mathbf{F}\,\text{body}(r_3)\}$$

We will spare the reader with the remaining nogoods in Λ_Π, that are obtained from all other subsets of $\text{atom}(\Pi)$.

Reconsider the assignments from Example 2.12, all of which are solutions for Δ_Π. However, observe that $\lambda^r_{\mathrm{ES}_\Pi(\{r,s\})} \subseteq \mathbf{A}_2$ and $\lambda^s_{\mathrm{ES}_\Pi(\{r,s\})} \subseteq \mathbf{A}_2$, i.e., above nogoods are violated w.r.t. $\mathbf{A}_2$. In particular, $\mathbf{A}_2$ is no solution for Λ_Π. Hence, by Theorem 2.4, $X_2 = \mathbf{A}_2^{\mathbf{T}} \cap \mathrm{atom}(\Pi)$ is no answer set of Π. On the other hand, $\mathbf{A}_1$ and $\mathbf{A}_3$ are solutions for Λ_Π. Hence, by Theorem 2.4, $X_1 = \mathbf{A}_1^{\mathbf{T}} \cap \mathrm{atom}(\Pi)$ and $X_3 = \mathbf{A}_3^{\mathbf{T}} \cap \mathrm{atom}(\Pi)$ are answer sets of Π. Indeed, they are (cf. Example 2.11).

The worst-case exponential number of loops in a program Π makes an explicit representation of Λ_Π infeasible in general (Lifschitz and Razborov, 2006). Most modern ASP solvers therefore encode loop nogoods on demand, for instance, whenever a loop nogood is conflicting (cf. Lin and Zhao, 2002; Giunchiglia et al., 2006; Gebser et al., 2007a).

Janhunen and Niemelä (2011) have proposed an alternative approach for characterising answer sets, centred around program transformations that only require sub-quadratic space in the size of the original program. The idea is based on *level rankings* (Niemelä, 2008), similar to level numberings (Janhunen, 2004), i.e., to map an atom p to a natural number (its level rank). We shall see that a supported model X of Π is an answer set of Π if and only if there exists a level ranking such that each atom $p \in X$ is in the head of an applicable rule whose atoms in its positive body have level ranks that are smaller that that of p.

We here consider Janhunen and Niemelä's level mapping transformation with weak ranking constraints.

Definition 2.8: Level Mapping of a Program (Janhunen and Niemelä, 2011)

For a program Π, the *level mapping* transformation, denoted by $\mathrm{LP2LP}(\Pi)$, is the program defined as follows:

$$\mathrm{LP2LP}(\Pi) = \bigcup_{\text{SCC } \mathsf{G} \text{ of } \mathrm{DG}^+(\Pi)} \mathrm{LP2LP}_\Pi(V(\mathsf{G}) \cap \mathrm{atom}(\Pi))$$

Observe that the level mapping exploits the SCCs of the positive dependency graph of Π. To establish compatibility, define $\mathrm{LP2LP}_\Pi(\emptyset) = \emptyset$. Then, in case of a singleton $V(\mathsf{G}) \cap \mathrm{atom}(\Pi) = \{p\}$, define

$$\mathrm{LP2LP}_\Pi(\{p\}) = \{r \in \Pi_{\{p\}} \mid r \not\in \mathrm{body}(r)^+\}.$$

For any set of atoms $S \subseteq \mathrm{atom}(\Pi)$ such that $|S| > 1$, let $n = \lceil \log_2 |S| \rceil$, and define

$$\mathrm{LP2LP}_\Pi(S) = \bigcup_{p \in S} \mathrm{LP2LP}_{\Pi,S}(p)$$

where $\mathrm{LP2LP}_{\Pi,S}(p)$ is constructed as follows:

1. Introduce a new atom of the form $\mathbf{bt}(r)$ for each rule $r \in \Pi_{\{p\}}$ to access its applicability, and split r into two rules

$$p \leftarrow \mathbf{bt}(r) \qquad\qquad \mathbf{bt}(r) \leftarrow \mathrm{body}(r).$$

2. For each rule $r \in \Pi_{\{p\}}$ such that $\mathrm{body}(r)^+ \cap S = \emptyset$, represent external support through

$$\mathbf{ext}(p) \leftarrow \mathbf{bt}(r)$$

where $\mathbf{ext}(p)$ is a new atom to represent whether p is externally supported.

3. To achieve a binary representation of the level rank of p, introduce new atoms of the form $\mathbf{lvl}(p)_i$ where $1 \leq i \leq n$. By convention, $\mathbf{lvl}(p)_1$ is the most significant bit. The following rules encode the level rank for p, and set it to 0 if p is false or externally supported:

$$\begin{aligned}
\mathbf{lvl}(p)_i &\leftarrow not\ \overline{\mathbf{lvl}}(p)_i \\
\overline{\mathbf{lvl}}(p)_i &\leftarrow not\ \mathbf{lvl}(p)_i \\
&\leftarrow \mathbf{lvl}(p)_i,\ not\ p \\
&\leftarrow \mathbf{lvl}(p)_i,\ p,\ \mathbf{ext}(p)
\end{aligned}$$

4. For each rule $r \in \Pi_{\{p\}}$ such that $\mathrm{body}(r)^+ \cap S = \{p_1, \ldots, p_m\} = \emptyset$, represent *internal* (non-external) support through

$$\mathbf{int}(p) \leftarrow \mathbf{bt}(r),\ \mathbf{lt}(p_1, p)_1, \ldots,\ \mathbf{lt}(p_m, p)_1,\ not\ \mathbf{ext}(p)$$

where $\mathbf{int}(p)$ is a new atom to represent whether p is internally supported. Intuitively, the above rule encodes that the atom p is internally supported if the level rank of each atom in its body sharing the same SCC is smaller than the level rank of p, and p is not externally supported. The new atoms of the form $\mathbf{lt}(q, p)_i$ represent that the binary encoded number $\mathbf{lvl}(q)_i \ldots \mathbf{lvl}(q)_n$ (i.e., from the level rank of atom q) is smaller than the binary encoded number $\mathbf{lvl}(p)_i \ldots \mathbf{lvl}(p)_n$ (i.e., from the level rank of atom p).

This is encoded by the set of rules

$$\mathbf{lt}(p_j, p)_i \leftarrow \mathbf{bt}(r),\ \mathbf{lvl}(p)_i,\ not\ \mathbf{lvl}(p_j)_i$$

where $1 \leq i \leq n$ and $1 \leq j \leq m$, and the set of rules

$$\mathbf{lt}(p_j, p)_i \leftarrow \mathbf{bt}(r),\ \mathbf{lt}(p_j, p)_{i+1},\ not\ \mathbf{lvl}(p)_i,\ not\ \mathbf{lvl}(p_j)_i$$
$$\mathbf{lt}(p_j, p)_i \leftarrow \mathbf{bt}(r),\ \mathbf{lt}(p_j, p)_{i+1},\ \mathbf{lvl}(p)_i,\ \mathbf{lvl}(p_j)_i$$

where $1 \leq i < n$ and $1 \leq j \leq m$.

5. Finally, it is required that if p is true, then it must be externally or internally supported:

$$\leftarrow p,\ not\ \mathbf{ext}(p),\ not\ \mathbf{int}(p).$$

Observe that the asymptotic space complexity of LP2LP(Π) is $\mathcal{O}(|\Pi| \log_2 |\Pi|)$. Janhunen and Niemelä have shown that the supported models of LP2LP(Π) correspond to the answer sets of Π.

Theorem 2.5: Models of the Level Mapping (Janhunen and Niemelä, 2011)

Let Π be a program. The supported models of LP2LP(Π) correspond many-to-one to the answer sets of Π. In particular, if X is a supported model of LP2LP(Π), then $X \cap \text{atom}(\Pi)$ is an answer set of Π.

We demonstrate the level mapping transformation in the following example.

Example 2.18

Reconsider the program Π from Example 2.11. Its transformation LP2LP(Π) is given through the following set of rules:

$$\begin{array}{ll}
p \leftarrow not\ q & \\
q \leftarrow not\ p & \\
r \leftarrow \mathbf{bt}(r_3) & \mathbf{bt}(r_3) \leftarrow p \\
r \leftarrow \mathbf{bt}(r_4) & \mathbf{bt}(r_4) \leftarrow s \\
s \leftarrow \mathbf{bt}(r_5) & \mathbf{bt}(r_5) \leftarrow r \\
\mathbf{ext}(r) \leftarrow \mathbf{bt}(r_3) & \\
\mathbf{lvl}(r)_1 \leftarrow not\ \overline{\mathbf{lvl}}(r)_1 & \mathbf{lvl}(s)_1 \leftarrow not\ \overline{\mathbf{lvl}}(s)_1 \\
\overline{\mathbf{lvl}}(r)_1 \leftarrow not\ \mathbf{lvl}(r)_1 & \overline{\mathbf{lvl}}(s)_1 \leftarrow not\ \mathbf{lvl}(s)_1
\end{array}$$

$$\leftarrow \mathbf{lvl}(r)_1,\ not\ r \qquad \leftarrow \mathbf{lvl}(r)_1,\ r,\ \mathbf{ext}(r)$$

$$\leftarrow \mathbf{lvl}(s)_1,\ not\ s \qquad \leftarrow \mathbf{lvl}(s)_1,\ s,\ \mathbf{ext}(s)$$

$$\mathbf{int}(r) \leftarrow \mathbf{bt}(r_4),\ \mathbf{lt}(s,r)_1,\ not\ \mathbf{ext}(r)$$

$$\mathbf{lt}(s,r)_1 \leftarrow \mathbf{bt}(r_4),\ \mathbf{lvl}(r)_1,\ not\ \mathbf{lvl}(s)_1$$

$$\mathbf{int}(s) \leftarrow \mathbf{bt}(r_5),\ \mathbf{lt}(r,s)_1,\ not\ \mathbf{ext}(s)$$

$$\mathbf{lt}(r,s)_1 \leftarrow \mathbf{bt}(r_5),\ \mathbf{lvl}(s)_1,\ not\ \mathbf{lvl}(r)_1$$

$$\leftarrow r,\ not\ \mathbf{int}(r),\ not\ \mathbf{ext}(r)$$

$$\leftarrow s,\ not\ \mathbf{int}(s),\ not\ \mathbf{ext}(s)$$

Verify that

$$X_1' = \{p, r, s, \mathbf{bt}(r_3), \mathbf{bt}(r_5), \mathbf{bt}(r_4), \mathbf{ext}(r), \mathbf{int}(s), \mathbf{lt}(r,s)_1, \mathbf{lvl}(s)_1\}, \text{ and}$$
$$X_3' = \{q\}$$

are the supported models of LP2LP(Π), and the projections $X_1' \cap \text{atom}(\Pi) = X_1$ and $X_3' \cap \text{atom}(\Pi) = X_3$ are the answer sets of Π. Hence, in our example, the level mapping transformation achieves a one-to-one correspondence between the answer sets of Π and the supported models of LP2LP(Π).

Janhunen and Niemelä also provide an encoding that achieves a one-to-one correspondence in general, using strong ranking constraints. For our purposes, however, the weak ranking constraints are sufficient to study the limits of the level mapping transformation.

Then, by Theorem 2.2, the answer sets of a program Π are given through solutions for $\Delta_{\text{LP2LP}(\Pi)}$, and therefore, the combination of search and UP provides a decision engine, i.e., CDNL, without need for checking unfounded set criteria.

Choice Rules, Cardinality Constraint Rules, and Integrity Constraints

We also consider some useful and widely used extension to programs: choice rules, cardinality constraint rules, and integrity constraints. Their semantics are given through program transformations that introduce additional atoms (cf. Simons et al., 2002). A *choice rule* has the form

$$\{h_1, \ldots, h_k\} \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n \tag{2.2}$$

and, if it is applicable w.r.t. some set of atoms X, it supports the atoms in an arbitrary subset of $\{h_1, \ldots, h_k\}$ w.r.t. X. In this thesis, we will see a rule of the form (2.2)

as a shorthand for

$$h_i \leftarrow p_1, \ldots, p_m, not\ \ p_{m+1}, \ldots,\ not\ p_n,\ not\ \overline{h_i}$$
$$\overline{h_i} \leftarrow not\ h_i$$

where $1 \leq i \leq k$, and $\overline{h_i}$ are new atoms. An *integrity constraint* is of the form

$$\leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n \tag{2.3}$$

i.e., an abbreviation for a rule with an unsupported head. Hence, a rule r of the form (2.3) must not be applicable w.r.t. any model of a program that includes r. It can be transformed into a rule of the form (2.1) as follows:

$$p_0 \leftarrow p_1, \ldots, p_m,\ not\ p_{m+1}, \ldots,\ not\ p_n,\ not\ p_0$$

The condition encoded by an integrity constraint, however, can be represented very efficiently by a single nogood, i.e., for an integrity constraint of the form (2.3), the nogood $\{\mathbf{T}p_1, \ldots, \mathbf{T}p_m, \mathbf{F}p_{m+1}, \ldots, \mathbf{F}p_n\}$. A *cardinality constraint rule* is of the following form:

$$p_0 \leftarrow k\{p_1, \ldots, p_n\} \tag{2.4}$$

It can be transformed into $\binom{n}{k}$ rules with head p_0, where for each k-elementary subset $\beta \subseteq \{p_1, \ldots, p_n\}$ there is a rule r with $\text{body}(r) = \beta$.

Alternatively, Simons et al. (2002) have provided a transformation that needs just $\mathcal{O}(nk)$ rules. The transformation, referred to as sequential counting (cf. Sinz, 2005), introduces additional atoms of the form $\mathbf{count}(i, j)$ to represent that at least j of the atoms with index $\geq i$ are included in a model. Then, a cardinality constraint rule of the form (2.4) can be transformed into the following set of rules:

$$p_0 \leftarrow \mathbf{count}(1, k) \tag{2.5}$$
$$\mathbf{count}(i, 1) \leftarrow p_i \tag{2.6}$$
$$\mathbf{count}(i, j) \leftarrow \mathbf{count}(i+1, j) \tag{2.7}$$
$$\mathbf{count}(i, j+1) \leftarrow p_i,\ \mathbf{count}(i+1, j) \tag{2.8}$$

where $1 \leq i \leq n$ and $1 \leq j \leq k$. Sinz has shown that UP on an encoding that uses sequential counting prunes all values.

Theorem 2.6: Boolean Cardinality via Unit Propagation (Sinz, 2005)

UP on (2.5–2.8) prunes all values in $\mathcal{O}(k)$ time down any branch of the search tree.

The key argument in the proof is that all nogoods represented by (2.5–2.8) are Horn-style, and therefore, can be solved in linear time via UP.

Another alternative transformation was put forward by Bomanson and Janhunen (2013), using merging and sorting constructions. Modern ASP solvers also incorporate specialised algorithms for handling cardinality constraints. In this thesis, we use both, the $\mathcal{O}(nk)$ transformation and specialised algorithms where appropriate. When $k = 2$ and space complexity is no issue, we even use the $\binom{n}{k} = \mathcal{O}(n^2)$ transformation. Sometimes, we use cardinality constraint rules without head, i.e., representing an unsupported head, of the following form:

$$\leftarrow k\{p_1, \ldots, p_n\} \tag{2.9}$$

Intuitively, such rule puts a restriction on the number of atoms allowed in the model. Note that, in practice, cardinality constraint rules also admit default literals. Other forms of aggregations are also common in ASP (Niemelä et al., 1999; Pelov, 2004; Gebser et al., 2009b; Faber et al., 2011). We here limit ourselves to the above concepts as they are expressive enough for our purposes.

Programs with Externals

In order to study the effect of ASP inference on sub-programs, or more generally, programs that integrate information from external sources, we use programs with external atoms. Formally, a *program with externals* $\mathcal{E}$ is a program Π over an alphabet distinguishing regular atoms $\mathcal{P}$ and external atoms $\mathcal{E}$, such that $\text{head}(r) \in \mathcal{P}$ for each $r \in \Pi$. In other words, external atoms do not appear in the head of any rule, but may provide information that determines the applicability of certain rules in the program.

Definition 2.9: Pre-Evaluation of a Program with Externals

Let Π be a program with externals $\mathcal{E}$, and $X \subseteq \mathcal{E}$. The *pre-evaluation* of Π w.r.t. X, denoted by $\Pi(X)$, is

$$\begin{aligned}\Pi(X) = \{\text{head}(r) \leftarrow &\{p \mid p \in \text{body}(r)^+ \setminus \mathcal{E}\} \cup \{not\ p \mid p \in \text{body}(r)^- \setminus \mathcal{E}\} \\ &\mid r \in \Pi, \text{body}(r)^+ \cap \mathcal{E} \subseteq X,\ \text{body}(r)^- \cap \mathcal{E} \cap X = \emptyset\}\end{aligned}$$

We obtain $\Pi(X)$ by removing all rules from Π whose positive body contains an external atom that is not in X or whose negative body contains an atom from X, and then dropping all (possibly default negated) external atoms from the remaining rules. Observe that any pre-evaluation of a program with externals is always a program without externals.

We now turn to splitting a program into sub-programs. Following Lifschitz and Turner (1994), we define splitting sets.

Definition 2.10: Splitting Set

Let Π be a program over $\mathscr{P}$. A *splitting set* for a program Π is a set $\mathscr{E} \subseteq \mathscr{P}$ if $\text{head}(r) \in \mathscr{E}$ then $\text{body}(r)^+ \cup \text{body}(r)^- \subseteq \mathscr{E}$ for each $r \in \Pi$.

Observe that, if $\mathscr{E}$ is a splitting set of Π, it *splits* Π into a program $\Pi_{\mathscr{E}}$ over $\mathscr{E}$ and a program $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ with externals $\mathscr{E}$.

There are two trivial splitting sets for a program Π, one being $\mathscr{E} = \emptyset$ and the other one being $\mathscr{E} = \text{atom}(\Pi)$. In either case, Π is split into $\emptyset$ and Π. The following is an example for a non-trivial splitting set.

Example 2.19

Reconsider the program Π over $\mathscr{P} = \{p, q, r, s\}$ from Example 2.11. Verify that $\mathscr{E} = \{p, q\}$ is a splitting set for Π. In particular, $\mathscr{E}$ splits the program Π into $\Pi_{\mathscr{E}}$, given by the rules r_1 and r_2, and the program $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ with externals $\mathscr{E}$, consisting of the rules r_3, r_4 and r_5:

$$\Pi_{\mathscr{E}} = \left\{ \begin{array}{ll} r_1: & p \leftarrow not\ q \\ r_2: & q \leftarrow not\ p \end{array} \right\} \text{ and } \Pi_{\mathscr{P}\setminus\mathscr{E}} = \left\{ \begin{array}{ll} r_3: & r \leftarrow p \\ r_4: & r \leftarrow s \\ r_5: & s \leftarrow r \end{array} \right\}.$$

Lifschitz and Turner put forward the following proposition, showing that splitting sets can be used to simplify the task of computing answer sets of a program.

Theorem 2.7: Splitting Set Theorem (Lifschitz and Turner, 1994)

Let Π be a program over $\mathscr{P}$, $\mathscr{E}$ a splitting set for Π, and $X \subseteq \mathscr{P}$. The set X is an answer set of Π if and only if $X = Y \cup Z$ such that Y is an answer set of $\Pi_{\mathscr{E}}$ and Z is an answer set of $\Pi_{\mathscr{P}\setminus\mathscr{E}}(Y)$.

In other words, a splitting set $\mathscr{E}$ breaks the task of computing an answer set of Π into two tasks:

1. computing an answer set Y of the program $\Pi_{\mathscr{E}}$, and using Y as an input for

(read below)

2. computing an answer set of $\Pi(Y)$, i.e., the pre-evaluation of the program $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ with externals over $\mathscr{E}$ w.r.t. Y.

Example 2.20

Continuing from Example 2.19, verify that the sets $Y_1 = \{p\}$ and $Y_3 = \{q\}$ are answer sets of $\Pi_{\mathscr{E}}$. We obtain the pre-evaluation of $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. Y_1, by removing all rules from Π whose negative body contains an atom from Y_1, and then dropping all default literals associated to external atoms in $\mathscr{E}$ from the remaining rules. We proceed with Y_3 in a similar fashion. Hence, we have

$$\Pi_{\mathscr{P}\setminus\mathscr{E}}(Y_1) = \left\{ \begin{array}{c} r \leftarrow \\ r \leftarrow s \\ s \leftarrow r \end{array} \right\} \text{ and } \qquad \Pi_{\mathscr{P}\setminus\mathscr{E}}(Y_3) = \left\{ \begin{array}{c} r \leftarrow s \\ s \leftarrow r \end{array} \right\}.$$

Verify that the sets $Z_1 = \{r, s\}$ and $Z_3 = \emptyset$ are the only answer sets of $\Pi_{\mathscr{P}\setminus\mathscr{E}}(Y_1)$ and $\Pi_{\mathscr{P}\setminus\mathscr{E}}(Y_3)$, respectively. By the splitting set theorem, $X_1 = Y_1 \cup Z_1 = \{p, r, s\}$ and $X_3 = Y_3 \cup Z_3 = \{q\}$ are answer sets of Π. Indeed, they are (cf. Example 2.11).

It is also important to note that the splitting set theorem works bidirectionally. In particular, it allows to join a program with externals, and a program that encodes those externals into one program. Moreover, recursive application of the splitting set theorem allows for decomposing a program into, and composing a program from further (sub-) programs.

2.6 Constraint Answer Set Programming

Constraint answer set programming (CASP; Baselice et al., 2005) is a special case of CSP that naturally merges CP and ASP, while preserving the advantages of both approaches to modelling and solving CSP. For instance, CP supports global constraints, whilst ASP permits recursive definitions and offers default negation.

An elegant, general formalism to facilitate the modelling in CASP is constraint logic programming. It is a form of logic programming that abstracts from non-propositional constraints by incorporating constraint atoms. Each constraint atom p is identified with the corresponding constraint via the function constraint(p).

Definition 2.11: Constraint (Logic) Program

A *constraint (logic) program* is a quadruple $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$, where $(\mathsf{V}, \mathsf{D}, \mathsf{C})$ is a CSP and Π is a program with externals $\mathscr{C}$, the *constraint atoms*, such that $\mathsf{C} = \{\text{constraint}(p) \mid p \in \mathscr{C}\}$.

Hence, for any constraint program $\mathbb{P}$ as above, it holds that $\text{head}(r) \notin \mathscr{C}$ for each $r \in \Pi$. To improve the modelling convenience, however, we follow Gebser et al. (2009c) and view a rule r with $\text{head}(r) \in \mathscr{C}$ as the integrity constraint

$$\leftarrow \text{body}(r),\ not\ \text{head}(r).$$

To access the constraint atoms in $\mathscr{C}$ that are associated with a constraint c, we define

$$\text{atom}(c) = \{p \mid p \in \mathscr{C}, \text{constraint}(p) = c\}$$
$$\text{atom}(\mathsf{C}) = \bigcup_{c \in \mathsf{C}} \text{atom}(c)$$

where C is a set of constraints. Hence, a fundamental difference to CP is that, in CASP, every constraint $c \in \mathsf{C}$ is reified via $\text{atom}(c)$. Their truth value is determined by the conditions induced by Π and an assignment A to the variables in $\text{scope}(c)$. We expand the notion of answer sets to define a declarative semantics for constraint programs.

Definition 2.12: Constraint Answer Sets of a Constraint Program

Let $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ be a constraint program, $X \subseteq \text{atom}(\Pi)$ and A a total assignment to the variables in V.

Then, the pair (X, A) is a *constraint answer set* of $\mathbb{P}$ if $(A, \text{sat}_{\mathsf{C}}(A))$ is a configuration for the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$ and X is an answer set of $\Pi(\text{atom}(\text{sat}_{\mathsf{C}}(A)))$.

Note that the pre-evaluation of Π w.r.t. $\text{atom}(\text{sat}_{\mathsf{C}}(A))$ is precisely the *constraint reduct* of Gebser et al. (2009c).

Example 2.21

Consider the constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ where $\mathsf{V} = \{\mathsf{v}\}$, $\mathsf{D} = \{\{1,2,3\}\}$ with $\text{dom}(\mathsf{v}) = \{1,2,3\}$, $\mathsf{C} = \{\mathsf{v} \leq 2\}$, and Π is the program over atoms $\{p, q, r, s\}$ and constraint atoms $\{[\![\mathsf{v} \leq 2]\!]\}$ consisting of the following rules:

$$p \leftarrow not\ q \qquad r \leftarrow p \qquad s \leftarrow r,\ [\![\mathsf{v} \leq 2]\!]$$
$$q \leftarrow not\ p \qquad r \leftarrow s$$

The constraint atom $[\![\mathsf{v} \leq 2]\!]$ is associated with the constraint $\mathsf{v} \leq 2$, i.e., we

have constraint([[v ≤ 2]]) = v ≤ 2 and atom(v ≤ 2) = {[[v ≤ 2]]}.

Verify that the pair (X_1, A_1) is a constraint answer set of $\mathbb{P}$, where $X_1 = \{p, r, s\}$ and A_1 such that $A_1(\mathsf{v}) = 2$. In particular, we have that $(A_1, \{\mathsf{v} \leq 2\})$ is a configuration for $(\mathsf{V}, \mathsf{D}, \mathsf{C})$, and X_1 is an answer set of the pre-evaluation of Π w.r.t. $\mathrm{atom}(\mathrm{sat}_{\mathsf{C}}(A_1)) = \{[[\mathsf{v} \leq 2]]\}$ given through the following rules:

$$p \leftarrow not\ q \qquad r \leftarrow p \qquad s \leftarrow r$$
$$q \leftarrow not\ p \qquad r \leftarrow s$$

Similarly, verify that the pair (X_3, A_1) is a constraint answer set of $\mathbb{P}$ where $X_3 = \{q\}$. Now, consider the assignment A_2 with $A_2(\mathsf{v}) = 3$. Hence, $\mathrm{sat}_{\mathsf{C}}(A_2) = \emptyset$ and $(A_2, \emptyset)$ is a configuration for $(\mathsf{V}, \mathsf{D}, \mathsf{C})$. Then, the pre-evaluation of Π w.r.t. $\mathrm{atom}(\mathrm{sat}_{\mathsf{C}}(A_1)) = \emptyset$ is given through the following rules:

$$p \leftarrow not\ q \qquad r \leftarrow p$$
$$q \leftarrow not\ p \qquad r \leftarrow s$$

Verify that the sets X_3 and $\{p, r\}$ are the answer sets of $\Pi(\emptyset)$ and, therefore, the pairs (X_3, A_2) and $(\{p, r\}, A_2)$ are constraint answer sets of $\mathbb{P}$.

Hybrid Constraint Answer Set Solving

Observe that Definition 2.12 provides us with instructions on how to compute constraint answer sets of a constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$. In particular, the problem can be broken down into two tasks:

1. computing configurations for the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$, i.e., pairs $(A, \mathrm{sat}_{\mathsf{C}}(A))$, and
2. computing the answer sets of $\Pi(\mathrm{atom}(\mathrm{sat}_{\mathsf{C}}(A)))$, i.e., the pre-evaluation of Π w.r.t. $\mathrm{atom}(\mathrm{sat}_{\mathsf{C}}(A))$.

Hybrid approaches to constraint answer set solving employ a CP solver for the former, and an ASP solver for the latter.

Following the idea of satisfiability modulo theories (SMT; Nieuwenhuis et al., 2006), hybrid CASP systems hand the constraint program to an ASP solver which ignores the theory behind constraint atoms, and constructs a model $X \cup Y$ where X is an answer set of $\Pi(Y)$, and Y is an arbitrary choice of constraint atoms representing $\mathrm{atom}(\mathrm{sat}_{\mathsf{C}}(A))$. In turn, the existence of a configuration $(A, \mathrm{sat}_{\mathsf{C}}(A))$ for the underlying CSP must be verified by the CP system. Balduccini and Lierler (2013)

have studied and evaluated various coupling mechanisms between the ASP and CP system, including a *clear-box* approach that performs the verification on partially constructed model candidate. If a configuration of the CSP exists, we have a constraint answer set. Otherwise, the model candidate is rejected. Another candidate has to be determined, possibly under considerations from CONFLICTANALYSIS that preclude a similar model. The procedure is repeated until it finds a constraint answer set, or the ASP solver determines that the program has no (further) answer sets. In case of the latter, the original constraint program has no constraint answer set.

The CASP systems *adsolver* and *acsolver* (Mellarkod and Gelfond, 2008; Mellarkod et al., 2008), *clingcon* (Gebser et al., 2009c), and *ezcsp* (Balduccini, 2009) follow this hybrid approach. Whilst *clingcon* combines the conflict-driven ASP solver *clasp* (Gebser et al., 2007b) and the CP system *gecode*, *ezcsp* offers the programmer a choice between the ASP solvers *clasp*, *cmodels* (Giunchiglia et al., 2006), and *dlv* (Leone et al., 2006), and between the CP systems *sicstus* and *b-prolog* (Zhou, 2012). This illustrates great flexibility but also the black-box character of the *hybrid* approach. For a comparison of the proof-theoretic similarities and differences between the syntax and algorithms of above systems, we refer the reader to (Lierler, 2014). The *adsolver* and *acsolver* systems integrate a backtracking-search decision algorithm with a CP solver for difference constraints, but disregard conflict-driven techniques. An experimental analysis conducted by Gebser et al. (2009c) revealed that they do not scale well. On the other hand, *ezcsp* employs conflict-driven search during model generation through *clasp*, though it does not consider information from model candidates rejected by the CP solver. From the above CASP systems, only *clingcon* attempts to overcome this limitation. Accordingly, Balduccini and Lierler (2013) described the integration schema of *clingcon* as tighter than *ezcsp*'s *clear-box* approach. However, from our perspective, *clingcon* still treats its CP solver as an oracle that does not contribute any information about its propagation, e.g., to the CONFLICTANALYSIS process. As a workaround, recent advances in *clingcon* (Ostrowski and Schaub, 2012) apply indirect measures to learn from the underlying CSP by looking at the implication graph between constraint atoms.

Chapter 3

Well-founded Justification and Well-founded Domination

Many native ASP solvers exploit unfounded sets via WFN to determine consequences of a program, but disregard WFJ due to computational cost. In fact, the most efficient technique to compute WFJ known to date is a combination of failed-literal-detection and FL (cf. Gebser and Schaub, 2013). This has high polynomial costs in size of the program.

In this chapter, we make several fundamental and foundational contributions to the propagation of unfounded sets.

- We devise a method that approximates the consequences of WFJ. Our technique is based on a novel graph-representation of programs written in ASP, termed the support flowgraph. We show that the problem of finding all dominators in such graph, for which linear-time algorithms exist, can be used to approximate WFJ.

- Our domination-based characterisation of WFJ gives rise to new forms of unfounded set inference that we call well-founded domination (WFD) and loop domination (LD). WFD and LD allow for the computation of additional consequences of a program. In particular, they include atoms into an answer set in order to guarantee external support to already included atoms.

- While our method provides an approximation of WFJ and WFD, in general, we will show that their effects can be simulated for important classes of programs. Those include important REACHABILITY conditions which we will study later in this thesis.

We start by motivating and introducing the support flowgraph of a program. It is similar to the positive dependency graph but dynamic in nature as it considers assignments.

3.1 Cuts in the Support Flowgraph

The inclusion of a set of atoms into an answer set of a program must be justified by the existence of external support. External support is provided by at least one body belonging to a rule whose head is from the set, but whose positive part does not include any of its atoms, and whose positive and negative dependencies are respected, i.e., included, respectively, excluded from the answer set. In turn, each positive dependency itself must be justified by the existence of external support. Ultimately, support is provided by bodies with no positive dependency, for instance, bodies that belong to facts.

Hence, if support were a resource, bodies with no positive dependency would have direct access, and could provide it to the atoms in the heads of their corresponding rules. In turn, these atoms would provide support as a resource to bodies that positively depend on them.

In this section, we will elaborate on this intuition by taking a look at how support *flows* through a program. First, we will tighten the notion of positive dependency as we do not wish to allow just any atom from the positive part of a body to provide support. While it is clear that all dependencies of a body (positive and negative) have to be respected in order for the body to truly supply support to atoms, we here relax this criterion and allow only a few dependencies to single-handedly support an entire body. At the same time, however, we want to retain information about circular dependency. We do so by distinguishing between atoms that share with a body the same strongly connected component in the positive dependency graph, and those that do not.

Definition 3.1: Support Dependency

For a program Π and a body $\beta \in \text{body}(\Pi)$, the *support dependency* of β w.r.t. Π is given through the mapping $\text{sup-dep}_\Pi : \text{body}(\Pi) \mapsto 2^{\text{atom}(\Pi)}$ defined by

$$\text{sup-dep}_\Pi(\beta) = \begin{cases} \beta^+ & \text{if } \beta^+ \cap \text{scc}(\beta) = \emptyset \\ \beta^+ \cap \text{scc}(\beta) & \text{otherwise.} \end{cases}$$

Observe that $\text{sup-dep}_\Pi(\beta) \subseteq \beta^+$, i.e., every support dependency of β is also a positive dependency, and that $\text{sup-dep}_\Pi(\beta) = \emptyset$, i.e., β has no support dependency, if and only if β has no positive dependency. As we shall see, restricting the intake of support by a body to support dependencies has some interesting properties.

We now formalise our previous intuition in terms of a flowgraph representation of the program.

Definition 3.2: Support Flowgraph

For a program Π and an assignment $\mathbf{A}$, the *support flowgraph* of Π w.r.t. $\mathbf{A}$ is the flowgraph $\text{SFG}(\Pi, \mathbf{A}) = (V, E)$, where

- the set of vertices is

$$V = \text{atom}(\Pi) \cup (\text{body}(\Pi) \setminus \mathbf{A}^{\mathbf{F}}) \cup \{\top\},$$

- the new symbol $\top$ is the designated source vertex, and
- the set of edges is

$$\begin{aligned} E = \quad & \{(\beta, p) \mid p \in \text{atom}(\Pi),\ \beta \in \text{body}(p) \setminus \mathbf{A}^{\mathbf{F}}\} \\ & \cup \{(p, \beta) \mid \beta \in \text{body}(\Pi),\ p \in \text{sup-dep}_\Pi(\beta) \setminus \mathbf{A}^{\mathbf{F}}\} \\ & \cup \{(\top, \beta) \mid \beta \in \text{body}(\Pi) \setminus \mathbf{A}^{\mathbf{F}},\ \text{sup-dep}_\Pi(\beta) = \emptyset\}. \end{aligned}$$

In words, the support flowgraph of a program (w.r.t. some assignment) consists of a vertex for each atom and each non-false body, and the source vertex. Furthermore, for each atom p, there is an edge that directs to the vertex corresponding to p from each vertex corresponding to a non-false body that belongs to a rule with head p, while for each vertex corresponding a non-false body β, there is an edge in the support flowgraph that directs to β from each vertex corresponding to a non-false support dependency of β. If β does not have any positive dependency, however, then there is an edge directed from the source vertex $\top$ to β.

We establish the following conventions: Vertices that correspond to atoms are referred to as *atom nodes*, and vertices that correspond to bodies are referred to as *body nodes*. We will also identify the atom nodes and body nodes in a support flowgraph with the corresponding atoms and bodies from the program.

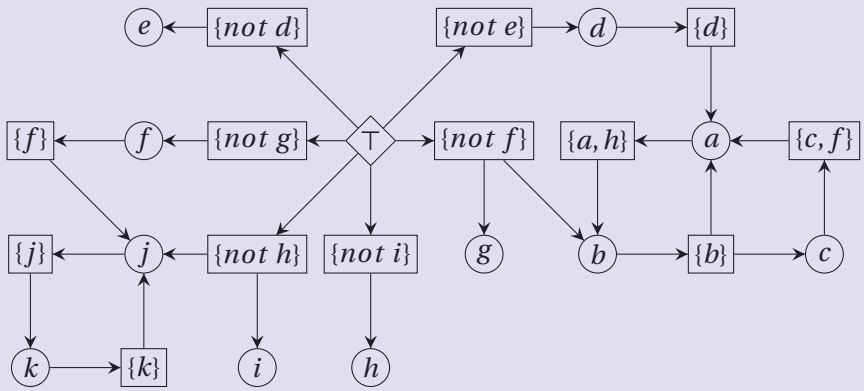
We illustrate the support flowgraph of a program in the following example.

Example 3.1

Consider the program Π with the following set of rules:

$$
\begin{array}{llllll}
a \leftarrow b & a \leftarrow c, f & a \leftarrow d & b \leftarrow a, h & b \leftarrow not\ f & c \leftarrow b \\
d \leftarrow not\ e & e \leftarrow not\ d & f \leftarrow not\ g & g \leftarrow not\ f & h \leftarrow not\ i & i \leftarrow not\ h \\
j \leftarrow f & j \leftarrow k & j \leftarrow not\ h & k \leftarrow j & &
\end{array}
$$

The support flowgraph of Π w.r.t. the empty assignment $\emptyset$, SFG($\Pi, \emptyset$), looks as follows. We use circle, rectangle, and diamond shapes to visually support the difference between vertices that stem from atoms and bodies, and the source.



Observe that there is an edge directed from the source vertex $\top$ to each non-false body that has no positive dependency. The other vertices and edges form a sub-graph similar to DG$^+(\Pi)$ with the exception of the edges from f to $\{c, f\}$, and from h to $\{a, h\}$. Those are missing in SFG($\Pi, \emptyset$) as $f \notin$ sup-dep$_\Pi(\{c, f\})$ and $h \notin$ sup-dep$_\Pi(\{a, h\})$.

A distinguished feature of the support flowgraph over related graph representations of a program, like the positive dependency graph, is that it is a dynamic rather than a static structure, i.e., it changes with the assignment. In every support flowgraph SFG(Π,**A**), however, we can observe a range of properties. They hold by construction and are easy to verify.

– Every edge that is adjacent to an atom node is adjacent from a body node, and either all edges adjacent to a body node are adjacent from atom nodes or there is a single edge that is adjacent to the body node, i.e., adjacent from the source vertex.

- There is no edge directed from or to a vertex corresponding to an atom or a body in $\mathbf{A}^{\mathbf{F}}$. For every cut c in $\mathrm{SFG}(\Pi, \mathbf{A})$ holds that $\mathrm{front}(\mathsf{c}) \cap \mathbf{A}^{\mathbf{F}} = \emptyset$.

- If $\mathbf{A}$ is body-saturated then there is an edge adjacent to every non-false body, and only bodies that are non-false have a corresponding vertex.

- Apart from the source vertex $\top$ and edges adjacent from $\top$, the support flowgraph of a program is always a sub-graph of the program's positive dependency graph. Formally, for $\mathrm{SFG}(\Pi, \mathbf{A}) = (V, E)$ we have $(V \setminus \{\top\}, E \setminus \{(\top, \beta) \mid \beta \in \mathrm{body}(\Pi)\}) \subseteq \mathrm{DG}^{+}(\Pi)$.

- Every loop of $\mathrm{SFG}(\Pi, \mathbf{A})$ is a loop of $\mathrm{DG}^{+}(\Pi)$ and, in agreement with our previous intuition, the strongly connected components of $\mathrm{SFG}(\Pi, \emptyset)$ are precisely the strongly connected components of $\mathrm{DG}^{+}(\Pi)$.

- The computational complexity for constructing $\mathrm{SFG}(\Pi, \mathbf{A})$ from scratch is linear in the size of Π. The construction, however, can be made incremental w.r.t. the assignment, i.e., edges and vertices are removed down any branch of the search tree and re-inserted upon backtracking.

We will formally prove one property of the support flowgraph, though, to provide reassurance that our intuition from the start of this section is met, i.e., every atom that is externally supported has in fact access to support as a resource provided by the source vertex via edges in the support flowgraph.

Theorem 3.1

Given a program Π and a body-saturated, unfounded-free assignment $\mathbf{A}$. For every atom $p \in \mathrm{atom}(\Pi) \setminus \mathbf{A}^{\mathbf{F}}$ there is a path from $\top$ to p in $\mathrm{SFG}(\Pi, \mathbf{A})$.

Proof. Let $p \in \mathrm{atom}(\Pi) \setminus \mathbf{A}^{\mathbf{F}}$ and U be the union of atoms in all paths in $\mathrm{SFG}(\Pi, \mathbf{A})$ from an atom node to p. Since $\mathbf{A}$ is unfounded-free, we have that $\mathrm{ES}_{\Pi}(U) \setminus \mathbf{A} \neq \emptyset$, i.e., there is a rule $r \in \Pi$ such that $\mathrm{body}(r) \notin \mathbf{A}^{\mathbf{F}}$, $\mathrm{head}(r) \in U$, and $\mathrm{body}(r)^{+} \cap U = \emptyset$. By Definition 3.2, there is an edge that directs from $\mathrm{body}(r)$ to $\mathrm{head}(r)$, and by assumption, we have a path from $\mathrm{head}(r)$ to p. Hence, there is a path from $\mathrm{body}(r)$ to p. Since $\mathbf{A}$ is body-saturated, by Definition 3.1, either there is an edge directed to $\mathrm{body}(r)$ from each atom in $\text{sup-dep}_{\Pi}(\beta) \subseteq \mathrm{body}(r)^{+}$, or the only edge adjacent to $\mathrm{body}(r)$ is adjacent to the source vertex $\top$. However, as U contains the atoms of all paths in $\mathrm{SFG}(\Pi, \mathbf{A})$ from an atom node to p, and $\mathrm{body}(r)^{+} \cap U = \emptyset$, the latter holds. In conclusion, there is a path from $\top$ to p. □

To analyse the flow of support we make use of cuts.

Definition 3.3: Support Cut

For a program Π and an assignment $\mathbf{A}$, a cut $\mathsf{c} = (S, W)$ in SFG($\Pi, \mathbf{A}$) is a *support cut* if $\top \in S$, front(c) $\subseteq$ body(Π), and back(c) $\subseteq$ body(Π).

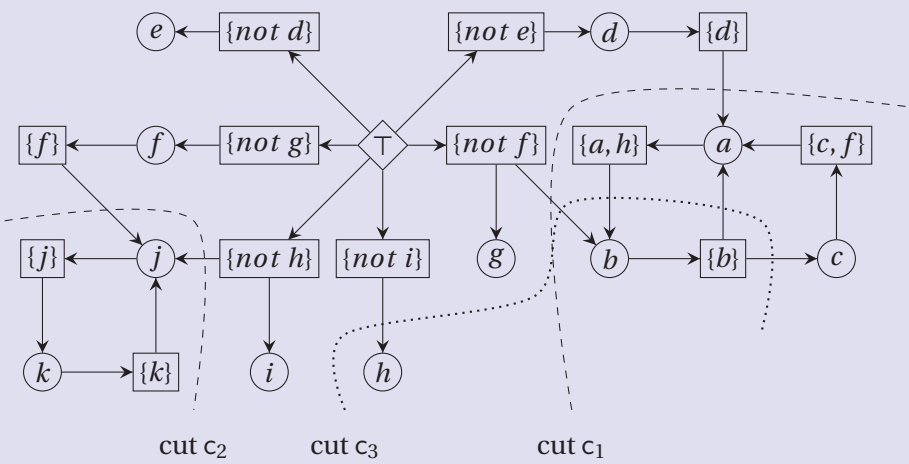
For a support cut $\mathsf{c} = (S, W)$ in SFG($\Pi, \mathbf{A}$), the condition front(c) $\subseteq$ body(Π) ensures that whenever a body is in W then all its support dependencies are in W, and back(c) $\subseteq$ body(Π) ensures that every body that has a support dependency in W is also in W. We will refer to vertices in S as being on the *support side* and to vertices in W as being on the *well-founded side* of c.

Example 3.2

Reconsider the program Π from Example 3.1. Verify that the following cuts

$$\begin{aligned}
\mathsf{c}_1 =(\{&\top, d, e, f, g, h, i, j, k, \{d\}, \{f\}, \{j\}, \{k\}, \{not\ d\}, \{not\ e\}, \{not\ f\},\\
&\{not\ g\}, \{not\ h\}, \{not\ i\}\},\ \{a, b, c, \{a, h\}, \{b\}, \{c, f\}\}),\\
\mathsf{c}_2 =(\{&\top, a, b, c, d, e, f, g, h, i, \{a, h\}, \{b\}, \{c, f\}, \{d\}, \{f\}, \{not\ d\}, \{not\ e\},\\
&\{not\ f\}, \{not\ g\}, \{not\ h\}, \{not\ i\}\},\ \{j, k, \{j\}, \{k\}\}), \text{ and}\\
\mathsf{c}_3 =(\{&\top, a, c, d, e, f, g, i, j, k, \{a, h\}, \{c, f\}, \{d\}, \{f\}, \{j\}, \{k\}, \{not\ d\}, \{not\ e\},\\
&\{not\ f\}, \{not\ g\}, \{not\ h\}, \{not\ i\}\},\ \{b, h, \{b\}\}),
\end{aligned}$$

all are support cuts in SFG($\Pi, \emptyset$). We illustrate c_1, c_2, and c_3 below.



Verify: $\{a, b, c\}$ is the set of atoms on the well-founded side of c_1, $\{j, k\}$ the one on the well-founded side of c_2, $\{b, h\}$ the one on the well-founded side of c_3.

We can make the following observations, setting the bodies that have an edge directed to an atom on the well-founded side of a cut in relation with their external support:

$$\text{front}(\mathsf{c}_1) = \{\{d\}, \{not\ f\}\} = \text{ES}_\Pi(\{a, b, c\}),$$
$$\text{front}(\mathsf{c}_2) = \{\{f\}, \{not\ h\}\} = \text{ES}_\Pi(\{j, k\}), \text{ and}$$
$$\text{front}(\mathsf{c}_3) = \{\{a, h\}, \{not\ i\}\} \supseteq \text{ES}_\Pi(\{b, h\}) = \{\{a, h\}\}.$$

In words, $\text{front}(\mathsf{c}_1)$ and $\text{front}(\mathsf{c}_2)$ represent external support of the atoms on their support side, while $\text{front}(\mathsf{c}_3)$ provides only an upper bound.

We now formalise our observation from Example 3.2 on the relationship between the bodies in $\text{front}(\mathsf{c})$ and the external support of the atoms on its well-founded side, i.e., every support cut separates a set of atoms from their external support.

Lemma 3.2: Separation Lemma

Given a program Π and a body-saturated assignment $\mathbf{A}$. If $\mathsf{c} = (S, W)$ is a support cut in $\text{SFG}(\Pi, \mathbf{A})$ then $\text{ES}_\Pi(W \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \text{front}(\mathsf{c})$.

Proof. Let $\mathsf{c} = (S, W)$ be a support cut in $\text{SFG}(\Pi, \mathbf{A})$. We verify that all non-false external support is included in $\text{front}(\mathsf{c})$. Let $r \in \Pi$ such that $\text{body}(r) \notin \mathbf{A}^\mathbf{F}$. By Definition 3.3, if $\text{head}(r) \in W$ then either $\text{body}(r) \in \text{front}(\mathsf{c})$ or $\text{body}(r) \in W$. If $\text{body}(r) \in \text{front}(\mathsf{c})$ then we simply declare that $\text{body}(r)$ may provide external support, so there is nothing left to prove. Suppose, however, that $\text{body}(r) \in W$. Then, again by Definition 3.3, there is no edge directed from the source vertex $\top$ to $\text{body}(r)$. Hence, it holds that $\text{sup-dep}_\Pi(\text{body}(r)) \neq \emptyset$, and since $\mathbf{A}$ is body-saturated we have $\text{sup-dep}_\Pi(\text{body}(r)) \subseteq W$. Therefore, $\text{body}(r)^+ \cap W \neq \emptyset$, i.e., if the body of a rule is on the well-founded side then it contains atoms from the well-founded side in its positive part. Therefore, it is not a member of the external support to the atoms in W. In conclusion, we get $\text{ES}_\Pi(W \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \text{front}(\mathsf{c})$, i.e., every external support to the set of atoms in W is a member of $\text{front}(\mathsf{c})$. □

Hence, the support flowgraph of a program can be used to extract information about non-false external support of a set of atoms, i.e., the set of bodies in $\text{front}(\mathsf{c})$ of a support cut c always provide an upper bound on the remaining external support of the atoms on the well-founded side of c. In the next section, we show how to exploit this property to compute consequences from WFJ.

3.2 Approximation of Well-Founded Justification

We now give special attention to support cuts that separate a set of atoms from a single external support, i.e., we want to identify body nodes that are in a domination relationship with a set of atoms.

Theorem 3.3: Approximation of WFJ

Given a program Π and a body-saturated, unfounded-free assignment $\mathbf{A}$. Let $U \subseteq \text{atom}(\Pi)$ and $r \in \Pi$. If $\text{body}(r)$ immediately dominates U in $\text{SFG}(\Pi, \mathbf{A})$ then $\text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F} = \{\text{body}(r)\}$.

Proof. Let $\text{body}(r)$ immediately dominate U in $\text{SFG}(\Pi, \mathbf{A})$. Then, there is a support cut $\mathsf{c} = (S, W)$ such that $\text{front}(\mathsf{c}) = \{\text{body}(r)\}$ and with all atoms in U on the well-founded site, i.e., $U = W \cap \text{atom}(\Pi)$. By the Separation Lemma, $\text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\text{body}(r)\}$. Since $\mathbf{A}$ is unfounded-free, it holds that $\text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F} \neq \emptyset$. We conclude $\text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F} = \{\text{body}(r)\}$. □

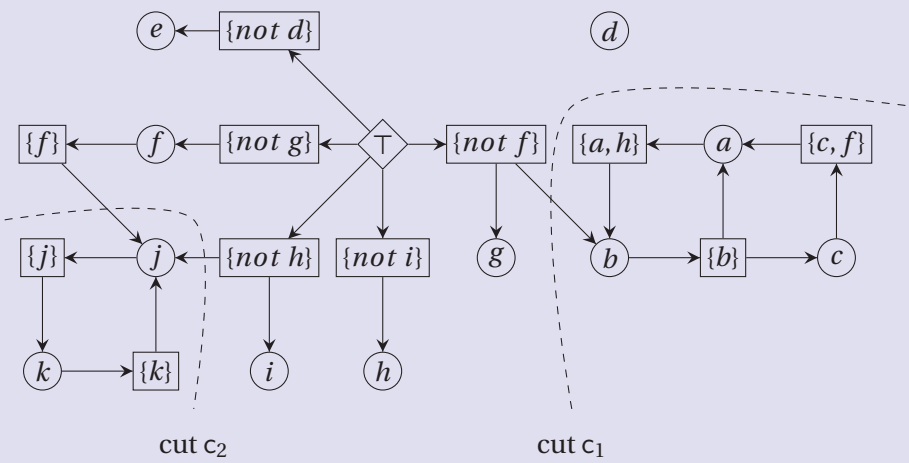
Hence, we can compute consequences from WFJ by searching for body nodes that are dominators in the support flowgraph of a program w.r.t. an assignment.

Example 3.3

Reconsider the program Π introduced in Example 3.1 with the support cuts c_1 and c_2 from Example 3.2 together with the assignment $\mathbf{A}$ given through

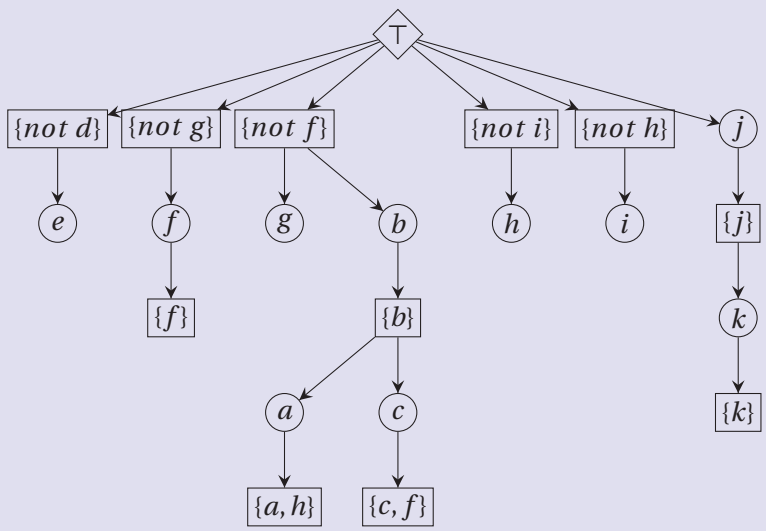
$$\mathbf{A} = \{\mathbf{T}a, \mathbf{F}d, \mathbf{T}e, \mathbf{T}j, \mathbf{T}k, \mathbf{F}\{d\}, \mathbf{T}\{j\}, \mathbf{T}\{k\}, \mathbf{T}\{not\ d\}, \mathbf{F}\{not\ e\}\}.$$

We illustrate the updated support flowgraph $\text{SFG}(\Pi, \mathbf{A})$ and both cuts below.



Verify that **A** is body-saturated and unfounded-free, and observe that $\{not\ f\}$, being the only body in front(c_1), immediately dominates all vertices on the well-founded side of c_1, i.e., the atoms in $\{a, b, c\}$.

The dominator tree of SFG(Π, **A**) looks like the following.



In fact, setting $\{not\ f\}$ to true is a consequence of applying BL or WFJ, i.e., WFJ[loop(Π)](Π, **A**) extends **A** by $\mathbf{T}\{not\ f\}$ since $ES_\Pi(\{a, b, c\}) \setminus \mathbf{A}^{\mathbf{F}} = \{\{not\ f\}\}$. On the other hand, there are two bodies in front(c_2), $\{f\}$ and $\{not\ h\}$, both provide external support to the set of atoms $\{j, k\}$ from the well-founded side of c_2.

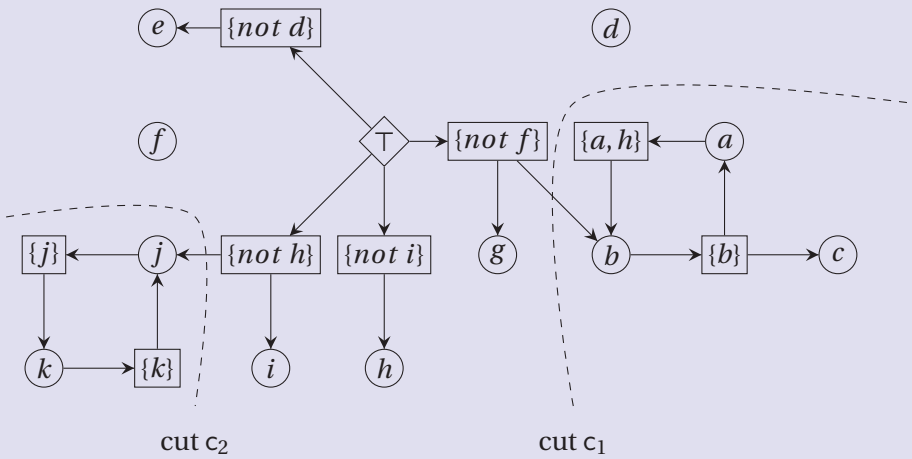
The application of WFJ can be interleaved with UP and WFN in order to compute further consequences from a program under an (extended) assignment.

Example 3.4: (continued)

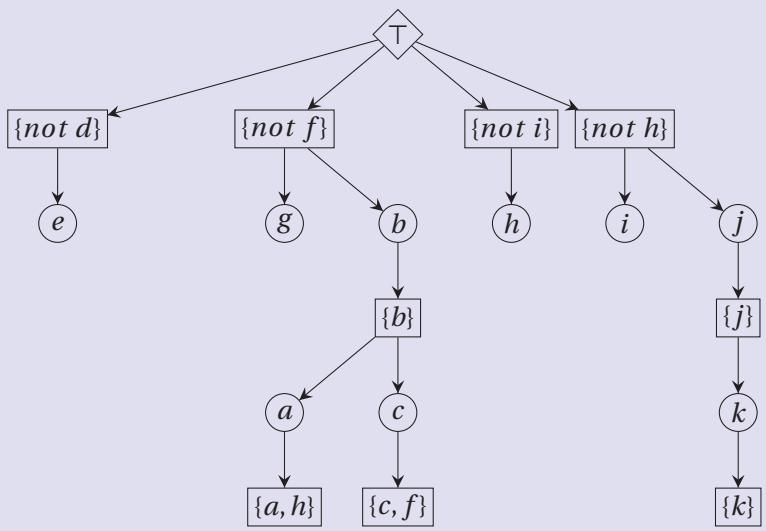
Continuing from where we have left Example 3.3, after extending **A** by $\mathbf{T}\{not\ f\}$, fixpoint operation of UP and WFN establishes the extended, body-saturated, unfounded-free assignment $\mathbf{A}'$ given through

$$\mathbf{A}' = \mathbf{A} \cup \{\mathbf{F}f, \mathbf{T}g, \mathbf{F}\{c, f\}, \mathbf{F}\{f\}, \mathbf{T}\{not\ f\}, \mathbf{F}\{not\ g\}\}.$$

The updated support flowgraph SFG(Π, $\mathbf{A}'$) along with the considered cuts is illustrated below.



The dominator tree of SFG($\Pi, \mathbf{A}'$), in comparison to the one of SFG($\Pi, \mathbf{A}$) has also changed, and looks as follows.



Observe that, having $\{f\}$ assigned false, the body $\{not\ h\}$ is the only remaining non-false external support of the set of atoms $\{j, k\}$. In fact, $\{not\ h\}$ dominates both j and k. This domination relationship coincides with computing consequences from BL or WFJ, as the application of BL or WFJ extends the assignment $\mathbf{A}'$ by $\mathbf{T}\{not\ h\}$. Formally, we have $\mathrm{WFJ}[\mathrm{loop}(\Pi)](\Pi, \mathbf{A}') = \mathbf{A}' \cup \{\mathbf{T}\{not\ h\}\}$.

Recall that finding all dominators in a flowgraph, e.g., by constructing the dominator tree, can be done in linear time w.r.t. the size of the program, and that

this computation can be made incremental, i.e., few dominators might be recomputed at any stage during search, subject to removal and re-insertion of vertices and edges. Hence, a domination-based method to approximate WFJ has a better asymptotic complexity than the best known method for computing consequences from WFJ (cf. Gebser and Schaub, 2013) by a linear factor, putting it on the same level of computational cost as WFN. As the combined run time complexity for unfounded set inference remains at $\mathcal{O}(|\Pi|)$, here is no worst case penalty to integrating our approximation into existing ASP solvers that already compute WFN.

3.3 Component-unary Programs

While a non-false body β being a dominator of some true atom is sufficient for $\mathbf{T}\beta$ being a consequence of WFJ, it is not a necessary condition. In fact, the converse of Theorem 3.3 does not hold in general. However, there is a class of programs for which it does, and then, all consequences from WFJ or BL can be computed by inspecting the dominator tree of a support flowgraph. As this can be done in linear time, our approach offers an algorithmic advance over the most efficient technique to compute WFJ or BL known to date (cf. Gebser and Schaub, 2013).

Definition 3.4: Unary and Component-unary Programs

A *component-unary (logic) program* is a program Π such that for every rule $r \in \Pi$ it holds that either $|\mathrm{body}(r)^+| \leq 1$ or $|\mathrm{body}(r)^+ \cap \mathrm{scc}(\mathrm{body}(r))| = 1$.
A *unary (logic) program* is a program Π such that for every rule $r \in \Pi$ it holds that $|\mathrm{body}(r)^+| \leq 1$.

It is easy to see that every unary program is a component-unary program. On the other hand, component-unary programs are much more general than unary programs, because unary programs cannot express conjunctive conditions (cf. Janhunen, 2000). A prominent example from the class of component-unary programs is discussed in Section 5.3. Similarly, however, component-unary programs cannot specify conjunctive conditions over atoms that do not share the same strongly connected component. This makes component-unary programs less expressive than (general) programs.

A key property of a program being component-unary is that, in a support flowgraph, there is exactly one edge adjacent to each body node.

Lemma 3.4

Given a component-unary program Π and an assignment $\mathbf{A}$. For every rule $r \in \Pi$ we have $|\text{sup-dep}_\Pi(\text{body}(r))| \leq 1$.

Proof. Let $r \in \Pi$. Since Π is component-unary, we have either $|\text{body}(r)^+| \leq 1$ or $|\text{body}(r)^+ \cap \text{scc}(\text{body}(r))| = 1$. The edges adjacent to a body node are determined by support dependency. By Definition 3.1, $\text{sup-dep}_\Pi(\text{body}(r)) \subseteq \text{body}(r)^+$. Hence, if $|\text{body}(r)^+| \leq 1$ then the claim holds trivially. On the other hand, if $|\text{body}(r)^+ \cap \text{scc}(\text{body}(r))| = 1$, meaning that there is an atom $p \in \text{atom}(\Pi)$ such that $\text{body}(r)^+ \cap \text{scc}(\text{body}(r)) = \{p\}$, then we get that also $\text{sup-dep}_\Pi(\beta) = \{p\}$, by Definition 3.1. In conclusion, we have $|\text{sup-dep}_\Pi(\text{body}(r))| \leq 1$. □

For what follows in the remainder of this chapter, we introduce a template for constructing a support cut from a set of atoms such that those atoms define the set of atoms on the well-founded side of the cut.

Definition 3.5: Support Cut w.r.t. a Set of Atoms

For a program Π and a set of atoms $U \subseteq \text{atom}(\Pi)$, the *support cut w.r.t. U*, denoted $\text{ES}_\Pi\text{-cut}(U)$, is the cut (S, W) where

- $W = U \cup \{\text{body}(r) \mid r \in \Pi,\ \text{sup-dep}_\Pi(\text{body}(r)) \cap U \neq \emptyset\}$, and
- $S = (\text{atom}(\Pi) \setminus W) \cup (\text{body}(\Pi) \setminus (W \cup \mathbf{A}^\mathbf{F})) \cup \{\top\}$.

In words, $\text{ES}_\Pi\text{-cut}(U)$ has all atoms from U and all bodies with a support dependency in U on the well-founded side. All other vertices of $\text{SFG}(\Pi, \mathbf{A})$, including $\text{ES}_\Pi(L)$ are on the support side of $\text{ES}_\Pi\text{-cut}(U)$. We can prove that $\text{ES}_\Pi\text{-cut}(U)$ is in fact a support cut in $\text{SFG}(\Pi, \mathbf{A})$.

Lemma 3.5

Given a component-unary program Π and an assignment $\mathbf{A}$. Let $U \subseteq \text{atom}(\Pi)$. $\text{ES}_\Pi\text{-cut}(U)$ is a support cut in $\text{SFG}(\Pi, \mathbf{A})$.

Proof. Let $\text{ES}_\Pi\text{-cut}(U) = (S, W)$. We already have $\text{back}(\text{ES}_\Pi\text{-cut}(U)) \subseteq \text{body}(\Pi)$ and $\top \notin \text{front}(\text{ES}_\Pi\text{-cut}(U))$, by Definition 3.5. In order to prove $\text{front}(\text{ES}_\Pi\text{-cut}(U)) \subseteq \text{body}(\Pi)$, it remains to demonstrate that $\text{front}(\text{ES}_\Pi\text{-cut}(U)) \cap \text{atom}(\Pi) = \emptyset$, i.e., there is no edge directed from an atom node in S to W. We show by proof of contradiction.

Suppose there is an atom $p \in \text{atom}(\Pi)$ such that $p \in \text{front}(\text{ES}_\Pi\text{-cut}(U))$. Then, by Definition 3.2, $p \notin \mathbf{A}^\mathbf{F}$ and there is a rule $r \in \Pi$ such that $\text{body}(r) \in W$ and

$p \in \text{sup-dep}_\Pi(\text{body}(r))$. Since Π is component-unary, we can apply Lemma 3.4 and get $|\text{sup-dep}_\Pi(\text{body}(r))| \leq 1$. In fact, we have $\text{sup-dep}_\Pi(\text{body}(r)) = \{p\}$. Therefore, $\text{sup-dep}_\Pi(\text{body}(r)) \cap U = \emptyset$. But then $\text{body}(r) \in S$, by Definition 3.5, contradicting the assumption. Hence, $\text{front}(\text{ES}_\Pi\text{-cut}(U)) \cap \text{atom}(\Pi) = \emptyset$. In conclusion, $\text{ES}_\Pi\text{-cut}(U)$ is a support cut in $\text{SFG}(\Pi, \mathbf{A})$. □

Example 3.5

Reconsider the program Π introduced in Example 3.1 with the support cuts c_1, c_2, and c_3 from Example 3.2. We can construct c_1, c_2, and c_3 as follows:

$$\mathsf{c}_1 = \text{ES}_\Pi\text{-cut}(\{a, b, c\})$$
$$\mathsf{c}_2 = \text{ES}_\Pi\text{-cut}(\{j, k\})$$
$$\mathsf{c}_3 = \text{ES}_\Pi\text{-cut}(\{b, h\})$$

For component-unary programs, the support cut w.r.t. a set of atoms captures the remaining non-false external support of that set if it forms a loop in the program.

Theorem 3.6

Given a component-unary program Π and a body-saturated $\mathbf{A}$. Let $L \in \text{loop}(\Pi)$. Then, we have that $\text{front}(\text{ES}_\Pi\text{-cut}(L)) = \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$.

Proof. Let $\text{ES}_\Pi\text{-cut}(L) = (S, W)$. We know from Lemma 3.5 that $\text{ES}_\Pi\text{-cut}(L)$ is a support cut in $\text{SFG}(\Pi, \mathbf{A})$. Moreover, by Definition 3.5, we have $L = W \cap \text{atom}(\Pi)$. Then, the Separation Lemma applies and we get $\text{front}(\text{ES}_\Pi\text{-cut}(L)) \supseteq \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$. In words, every external support of L is in $\text{front}(\text{ES}_\Pi\text{-cut}(L))$. To prove the theorem, it remains to show that also $\text{front}(\text{ES}_\Pi\text{-cut}(L)) \subseteq \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$. To begin with, we will demonstrate that for every rule $r \in \Pi$ with $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(L))$ it holds that $\text{body}(r) \in \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$. We show by proof of contradiction.

Suppose there is $r \in \Pi$ such that $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(L))$ and $\text{body}(r) \notin \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$. Since $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(L))$, by Definition 3.2, we have that $\text{body}(r) \notin \mathbf{A}^\mathbf{F}$ and $\text{head}(r) \in L$, and by Definition 3.3, we have $\text{sup-dep}_\Pi(\text{body}(r)) \cap L = \emptyset$. Now, since $\text{body}(r) \notin \mathbf{A}^\mathbf{F}$, $\text{head}(r) \in L$, and $\text{body}(r) \notin \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$, we get $\text{body}(r)^+ \cap L \neq \emptyset$, by definition of external support. Furthermore, from $L \in \text{loop}(\Pi)$, $\text{head}(r) \in L$, and $\text{body}(r)^+ \cap L \neq \emptyset$ we conclude $\text{body}(r)^+ \cap \text{scc}(\text{body}(r)) \neq \emptyset$. Therefore, by Definition 3.1, $\text{sup-dep}_\Pi(\text{body}(r)) \cap L \neq \emptyset$, and by Definition 3.5, $\text{body}(r) \in W$, contradicting the assumption that $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(L))$. Hence, for every rule $r \in \Pi$ with $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(L))$ it holds that $\text{body}(r) \in \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$.

Finally, since $\text{ES}_\Pi\text{-cut}(L)$ is a support cut in $\text{SFG}(\Pi, \mathbf{A})$, by Lemma 3.5, we have $\text{front}(\text{ES}_\Pi\text{-cut}(L)) \subseteq \text{body}(\Pi)$, and therefore, $\text{front}(\text{ES}_\Pi\text{-cut}(L)) \subseteq \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$. In conclusion, $\text{front}(\text{ES}_\Pi\text{-cut}(L)) = \text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F}$. □

Hence, for component-unary programs, we can exploit the domination relationship between body and atom nodes to compute all consequences from BL.

Theorem 3.7: BL for Component-unary Programs

Given a component-unary program Π and a body-saturated, unfounded-free assignment $\mathbf{A}$. Let $L \in \text{loop}(\Pi)$ and $r \in \Pi$ such that $\text{body}(r) \notin \mathbf{A}^\mathbf{F}$. Then, $\text{body}(r)$ immediately dominates L in $\text{SFG}(\Pi, \mathbf{A})$ if and only if $\text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F} = \{\text{body}(r)\}$.

Proof. The implication ($\Rightarrow$) holds by Theorem 3.3. It remains to show ($\Leftarrow$). To begin, let $\text{ES}_\Pi(L) \setminus \mathbf{A}^\mathbf{F} = \{\text{body}(r)\}$. By Lemma 3.5, $\text{ES}_\Pi\text{-cut}(L)$ is a support cut in $\text{SFG}(\Pi, \mathbf{A})$, and $\text{front}(\text{ES}_\Pi\text{-cut}(L)) = \{\text{body}(r)\}$ by Theorem 3.6. Hence, $\text{body}(r)$ immediately dominates L in $\text{SFG}(\Pi, \mathbf{A})$. □

While previous theorems apply to sets of atoms that are loops of a program, we now consider any set of atoms. For unary programs, the support cut w.r.t. a set of atoms captures remaining non-false external support.

Theorem 3.8

Given a unary program Π and a body-saturated assignment $\mathbf{A}$. Let $U \subseteq \text{atom}(\Pi)$. Then, we have that $\text{front}(\text{ES}_\Pi\text{-cut}(U)) = \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$.

Proof. Let $\text{ES}_\Pi\text{-cut}(U) = (S, W)$. We know from Lemma 3.5 that $\text{ES}_\Pi\text{-cut}(U)$ is a support cut in $\text{SFG}(\Pi, \mathbf{A})$. Moreover, by Definition 3.5, we have $U = W \cap \text{atom}(\Pi)$. Then, the Separation Lemma applies and we get $\text{front}(\text{ES}_\Pi\text{-cut}(U)) \supseteq \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$, i.e., every external support of U is in $\text{front}(\text{ES}_\Pi\text{-cut}(U))$. It remains to show that also $\text{front}(\text{ES}_\Pi\text{-cut}(U)) \subseteq \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$. To begin, we will demonstrate that for every rule $r \in \Pi$ with $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(U))$ it holds that $\text{body}(r) \in \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$.

We show by proof of contradiction. Suppose there is $r \in \Pi$ such that $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(U))$ and $\text{body}(r) \notin \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$. Since $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(U))$, by Definition 3.2, we have that $\text{body}(r) \notin \mathbf{A}^\mathbf{F}$ and $\text{head}(r) \in U$, and by Definition 3.3, $\text{sup-dep}_\Pi(\text{body}(r)) \cap U = \emptyset$. Moreover, since Π is unary, Lemma 3.4 applies, and we get $\text{sup-dep}_\Pi(\text{body}(r)) = \text{body}(r)^+$. From $\text{sup-dep}_\Pi(\text{body}(r)) \cap U = \emptyset$, we conclude $\text{body}(r)^+ \cap U = \emptyset$, and then, we have $\text{body}(r) \in \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$ contradicting the assumption that $\text{body}(r) \notin \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$. Hence, for every $r \in \Pi$ with $\text{body}(r) \in \text{front}(\text{ES}_\Pi\text{-cut}(U))$ it holds that $\text{body}(r) \in \text{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$.

Finally, since $\mathrm{ES}_\Pi\text{-cut}(U)$ is a support cut in $\mathrm{SFG}(\Pi, \mathbf{A})$, by Lemma 3.5, we have $\mathrm{front}(\mathrm{ES}_\Pi\text{-cut}(U)) \subseteq \mathrm{body}(\Pi)$, and therefore, $\mathrm{front}(\mathrm{ES}_\Pi\text{-cut}(U)) \subseteq \mathrm{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$. In conclusion, $\mathrm{front}(\mathrm{ES}_\Pi\text{-cut}(U)) = \mathrm{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F}$. □

Given this, for unary programs, we can guarantee that the domination relationship between body and atom nodes in the support flowgraph can be used to compute all consequences from WFJ.

Theorem 3.9: WFJ for Unary Programs

Given a unary program Π and a body-saturated, unfounded-free assignment $\mathbf{A}$. Let $U \subseteq \mathrm{atom}(\Pi)$ and $r \in \Pi$ such that $\mathrm{body}(r) \notin \mathbf{A}^\mathbf{F}$. The body $\mathrm{body}(r)$ immediately dominates U in $\mathrm{SFG}(\Pi, \mathbf{A})$ if and only if $\mathrm{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F} = \{\mathrm{body}(r)\}$.

Proof. The implication ($\Rightarrow$) holds by Theorem 3.3. It remains to show ($\Leftarrow$). To begin, let $\mathrm{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F} = \{\mathrm{body}(r)\}$. By Lemma 3.5, $\mathrm{ES}_\Pi\text{-cut}(U)$ is a support cut in $\mathrm{SFG}(\Pi, \mathbf{A})$, and $\mathrm{front}(\mathrm{ES}_\Pi\text{-cut}(U)) = \{\mathrm{body}(r)\}$ by Theorem 3.8. Hence, $\mathrm{body}(r)$ immediately dominates U in $\mathrm{SFG}(\Pi, \mathbf{A})$. □

In the sequel, we will demonstrate that BL can not be replaced by UP and WFN, even if level mapping transformations (Janhunen and Niemelä, 2011) are used.

So far, we have restricted our attention to body nodes that dominate a set of atom nodes. In principle, however, any type of vertex can be a (strict) dominator in the flowgraph of a program w.r.t. some assignment. We will address dominators that are atom nodes in the next section.

3.4 Well-Founded Domination

We define an atom-equivalent of WFJ, i.e., if a set of atoms U contains at least one true atom, then any atom that appears positively in all external support of U must likewise be true.

Definition 3.6: Well-Founded Domination

For a set of sets of atoms Ω, a program Π, and an assignment $\mathbf{A}$, the *well-founded domination* operator is defined as

$$\mathrm{WFD}[\Omega](\Pi, \mathbf{A}) = \begin{cases} \mathbf{A} \cup \{\mathbf{T}p\} & \text{if } U \cup \{p\} \in \Omega,\ U \cap \mathbf{A}^\mathbf{T} \neq \emptyset, \text{ and} \\ & \mathrm{ES}_\Pi(U) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\mathrm{body}(r) \mid r \in \Pi,\ p \in \mathrm{body}(r)^+\}, \\ \mathbf{A} & \text{otherwise.} \end{cases}$$

As with WFJ, we consider the two alternatives $\Omega = 2^{\text{atom}(\Pi)}$ (well-founded domination, WFD) and $\Omega = \text{loop}(\Pi)$ (loop domination, LD), i.e., all sets of atoms and loops from the program, respectively. It is easy to see that LD and WFD do not interfere with the completeness of any inference system for ASP: Since every set of atoms containing a true-assigned atom cannot be an unfounded set, at least one external support must hold. Hence, it is safe to infer all literals that are shared amongst all non-false external support. In particular, it is safe to infer the truth of all atoms shared amongst the positive of all non-false external support.

In this section, we will show how the support flowgraph of a program can be reused to also compute consequences of LD and WFD. We follow a strategy similar to the one from the previous sections, and start by defining a new form of cut, the atom cut.

Definition 3.7: Atom Cut

For a program Π and an assignment $\mathbf{A}$, a cut $\mathsf{c} = (S, W)$ in $\text{SFG}(\Pi, \mathbf{A})$ is an *atom cut* if $\top \in S$, $\text{front}(\mathsf{c}) \subseteq \text{atom}(\Pi)$, and $\text{back}(\mathsf{c}) \subseteq \text{body}(\Pi)$.

For an atom cut $\mathsf{c} = (S, W)$ in $\text{SFG}(\Pi, \mathbf{A})$, the conditions $\text{front}(\mathsf{c}) \subseteq \text{atom}(\Pi)$ and $\text{back}(\mathsf{c}) \subseteq \text{body}(\Pi)$ ensure that every body that has a support dependency in W or corresponds to a rule with a head in W is also in W. Hence, the distinguishing property of atom cuts is, in comparison to support cuts, that atom cuts have edges directed from atom nodes off the support side to body nodes on the well-founded side of the cut. Like with support cuts, we will refer to vertices in S as being on the support side and to vertices in W as being on the well-founded side of c.

Example 3.6

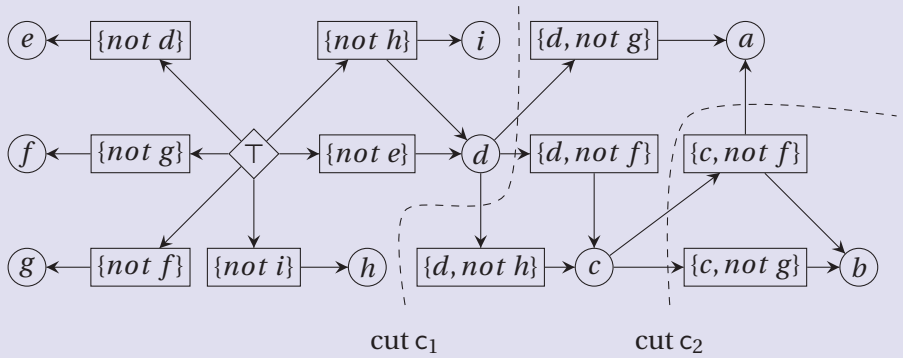
Consider the program Π with the following set of rules:

$$\begin{array}{lllll} a \leftarrow c, not\ f & a \leftarrow d, not\ g & b \leftarrow c, not\ f & b \leftarrow c, not\ g & c \leftarrow d, not\ f \\ c \leftarrow d, not\ h & d \leftarrow not\ e & d \leftarrow not\ h & e \leftarrow not\ d & f \leftarrow not\ g \\ g \leftarrow not\ f & h \leftarrow not\ i & i \leftarrow not\ h & & \end{array}$$

Verify that the two cuts c_1 and c_2 given through

$$\begin{aligned} \mathsf{c}_1 =& (\{\top, d, e, f, g, h, i, \{not\ d\}, \{not\ e\}, \{not\ f\}, \{not\ g\}, \{not\ h\}, \{not\ i\}\}, \\ & \{a, b, c, \{c, not\ f\}, \{c, not\ g\}, \{d, not\ f\}, \{d, not\ g\}, \{d, not\ h\}\}), \\ \mathsf{c}_2 =& (\{\top, a, c, d, e, f, g, h, i, \{d, not\ f\}, \{d, not\ g\}, \{d, not\ h\}, \{not\ d\}, \\ & \{not\ e\}, \{not\ f\}, \{not\ g\}, \{not\ h\}, \{not\ i\}\}, \{b, \{c, not\ f\}, \{c, not\ g\}\}, \end{aligned}$$

both are atom cuts in SFG($\Pi, \emptyset$). The support flowgraph SFG($\Pi, \emptyset$) along with c_1 and c_2 looks like below.



The set $\{a, b, c\}$ is the set of atoms on the well-founded side of c_1, and $\{b\}$ the one on the well-founded side of c_2. Verify that the atom in front(c_1) = $\{d\}$ appears positively in bodies from $ES_\Pi(\{a, b, c\}) = \{\{d, not\ f\}, \{d, not\ g\}, \{d, not\ h\}\}$, and that the one in front(c_2) = $\{c\}$ appears positively in bodies from $ES_\Pi(\{b\}) = \{\{c, not\ f\}, \{c, not\ g\}\}$.

Building on the observation from Example 3.6, we formalise an atom-equivalent of the Separation Lemma. It guarantees that every atom cut in SFG($\Pi, \mathbf{A}$) separates a set of atoms from the set of atoms that appear positively in its external support.

Lemma 3.10

Given a program Π and a body-saturated assignment $\mathbf{A}$. If $c = (S, W)$ is an atom cut in SFG($\Pi, \mathbf{A}$) then $ES_\Pi(W \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ \text{body}(r)^+ \cap \text{front}(c) \neq \emptyset\}$.

Proof. Let $c = (S, W)$ be an atom cut in SFG($\Pi, \mathbf{A}$). Let $B = W \cap \{\text{body}(r) \mid r \in \Pi,\ \text{sup-dep}_\Pi(\text{body}(r)) \cap \text{front}(c) \neq \emptyset\}$. In words, B is the set of bodies in W adjacent to an edge that directs from a vertex in front(c). In particular, for all $\beta \in B$ we have $\beta^+ \cap \text{front}(c) \neq \emptyset$. We construct a new cut $c' = (S', W')$ where $S' = S \cup B$ and $W' = W \setminus B$, i.e., all bodies in B are shifted from the well-founded side of c to the support side of c'. All other vertices, however, remain on their respective side, e.g., $W \cap \text{atom}(\Pi) = W' \cap \text{atom}(\Pi)$. In particular, $\text{back}(c') = \text{back}(c) \subseteq \text{body}(\Pi)$.

Next, recall that, by Definition 3.2, in a support flowgraph, all edges direct to body node from either an atom node or the source vertex, or direct body nodes

to atom. Hence, through shifting, we get $\text{front}(\mathsf{c}') \subseteq B \subseteq \text{body}(\Pi)$ and, therefore, c' is a support cut in $\text{SFG}(\Pi, \mathbf{A})$. By the Separation Lemma, $\text{ES}_\Pi(W' \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \text{front}(\mathsf{c}')$. Since $W \cap \text{atom}(\Pi) = W' \cap \text{atom}(\Pi)$, we get $\text{ES}_\Pi(W \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \text{front}(\mathsf{c}')$, and since $\text{front}(\mathsf{c}') \subseteq B$ we get $\text{ES}_\Pi(W \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ \text{sup-dep}_\Pi(\text{body}(r)) \cap \text{front}(\mathsf{c}) \neq \emptyset\}$. By Definition 3.1, $\text{sup-dep}_\Pi(\text{body}(r)) \subseteq \text{body}(r)^+$, and we conclude $\text{ES}_\Pi(W \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ \text{body}(r)^+ \cap \text{front}(\mathsf{c}) \neq \emptyset\}$. □

Hence, for an atom cut c in $\text{SFG}(\Pi, \mathbf{A})$, the atoms in $\text{front}(\mathsf{c})$ provide an upper bound on the atoms that appear positively in all external support of the atoms on the well-founded side of c.

However, we can only draw a meaningful conclusion if the atoms in $\text{front}(\mathsf{c})$ are shared amongst the positive dependency of all external support. Therefore, we restrict our attention to atom cuts with a single member in $\text{front}(\mathsf{c})$, i.e., dominators. This restriction guarantees that $\text{front}(\mathsf{c})$ represents the intersection of all external support, trivially, and allows us to approximate WFD easily.

Theorem 3.11: Approximation of WFD

Given a program Π and a body-saturated assignment $\mathbf{A}$. Let $U \subseteq \text{atom}(\Pi)$ and $p \in U \setminus \mathbf{A}^\mathbf{F}$. If p immediately dominates U in $\text{SFG}(\Pi, \mathbf{A})$ then $\text{ES}_\Pi(U \setminus \{p\}) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ p \in \text{body}(r)^+\}$.

Proof. In principle, the proof follows from Lemma 3.10.

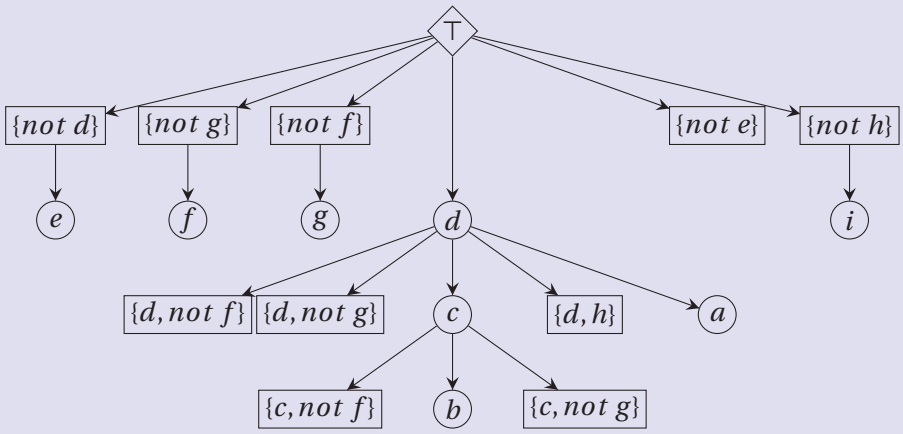
Let p immediately dominate U in $\text{SFG}(\Pi, \mathbf{A})$. Since $p \in U$, p immediately dominates $U \setminus \{p\}$. Then, there exists an atom cut $\mathsf{c} = (S, W)$ such that $\text{front}(\mathsf{c}) \subseteq \{p\}$ and $U \setminus \{p\} = W \cap \text{atom}(\Pi)$. By Lemma 3.10, $\text{ES}_\Pi(W \cap \text{atom}(\Pi)) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ \text{body}(r)^+ \cap \text{front}(\mathsf{c}) \neq \emptyset\}$ and therefore, $\text{ES}_\Pi(U \setminus \{p\}) \setminus \mathbf{A}^\mathbf{F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ p \in \text{body}(r)^+\}$. □

Example 3.7

Reconsider the program Π and atom cuts c_1 and c_2 from Example 3.6 together with the assignments $\mathbf{A} = \{\mathbf{T}a\}$ and $\mathbf{A}' = \{\mathbf{T}b\}$. The flowgraphs $\text{SFG}(\Pi, \mathbf{A})$ and $\text{SFG}(\Pi, \mathbf{A}')$ are the same as in the previous example.

Verify that $\mathbf{A}$ and $\mathbf{A}'$ are body-saturated, and observe that the atom d from $\text{front}(\mathsf{c}_1)$, being the only atom that appears positively in all external support of $\{a, b, c\}$, dominates all vertices on the well-founded side of c_1. Similarly, the atom c from $\text{front}(\mathsf{c}_2)$, being the only atom that appears positively in all exter-

nal support of $\{b\}$, dominates all vertices on the well-founded side of c_2. This is reflected in the dominator tree of SFG(Π,**A**) and SFG(Π,**A**$'$), as illustrated below.



Hence, given **A**, WFD sets d to true, i.e., WFD[atom(Π)](Π,**A**) extends **A** by **T**d. Similarly for **A**$'$, WFD[atom(Π)](Π,**A**$'$) non-deterministically extends **A**$'$ by **T**c and **T**d.

For component-unary programs, the domination relationship between atoms can be used to compute all consequences of LD.

Theorem 3.12: LD for Component-unary Programs

Given a component-unary program Π and a body-saturated assignment **A**. Let $L \in \text{loop}(\Pi)$ and $p \in L \setminus \mathbf{A^F}$. The atom p immediately dominates L in SFG(Π,**A**) if and only if $\text{ES}_\Pi(L \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ p \in \text{body}(r)^+\}$.

Proof. The implication ($\Rightarrow$) holds by Theorem 3.11. It remains to show ($\Leftarrow$) that if $\text{ES}_\Pi(L \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ p \in \text{body}(r)^+\}$ then p immediately dominates L in SFG(Π,**A**). Let $\text{ES}_\Pi(L \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ p \in \text{body}(r)^+\}$. By Lemma 3.5, $\text{ES}_\Pi\text{-cut}(L \setminus \{p\})$ is a support cut in SFG(Π,**A**), and by Theorem 3.6, $\text{front}(\text{ES}_\Pi\text{-cut}(L \setminus \{p\})) = \text{ES}_\Pi(L \setminus \{p\}) \setminus \mathbf{A^F}$. Hence, by assumption that $\text{ES}_\Pi(L \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{\text{body}(r) \mid r \in \Pi,\ p \in \text{body}(r)^+\}$, we have $\text{front}(\text{ES}_\Pi\text{-cut}(L \setminus \{p\})) \subseteq \{\text{body}(r) \mid r \in \Pi,\ p \in \text{body}(r)^+\}$. Next, let $\text{ES}_\Pi\text{-cut}(L \setminus \{p\}) = (S, W)$. We construct a new cut $\mathsf{c} = (S', W')$ where $S' = S \setminus \text{front}(\text{ES}_\Pi\text{-cut}(L \setminus \{p\}))$ and $W' = W \cup \text{front}(\text{ES}_\Pi\text{-cut}(L \setminus \{p\}))$, i.e., all non-false external support of $L \setminus \{p\}$ is shifted from the support side of $\text{ES}_\Pi\text{-cut}(L \setminus \{p\})$ to the well-founded side of c. All other vertices, however, remain on their respective side, e.g., $W \cap \text{atom}(\Pi) = W' \cap \text{atom}(\Pi)$. In particular, $\text{back}(\mathsf{c}) = \text{back}(\text{ES}_\Pi\text{-cut}(L \setminus \{p\})) \subseteq$

body(Π). Next, recall that, by Definition 3.2, in a support flowgraph, all edges direct to body node from either an atom node or the source vertex, or direct body nodes to atom. Hence, through shifting, we get front(c) ⊆ atom(Π), and therefore, c is an atom cut in SFG(Π, **A**).

It remains to show that front(c) = $\{p\}$. Let $r \in \Pi$ be a rule such that body$(r) \in$ front(ES$_\Pi$-cut$(L \setminus \{p\})$). By construction, we have $p \in$ body$(r)^+$. Given this, from $L \in$ loop(Π), head$(r) \in L$, and $p \in L$, we conclude body$(r)^+ \cap$ scc(body$(r)) \supseteq \{p\}$. Then, by Definition 3.1, we have sup-dep$_\Pi$(body$(r)) \supseteq \{p\}$. Since Π is component-unary, Lemma 3.4 applies. Hence, we have |sup-dep$_\Pi$(body(r))| ≤ 1, and can conclude that sup-dep$_\Pi$(body$(r)) = \{p\}$.

Therefore, c is an atom cut in SFG(Π, **A**) with atoms in $L \setminus \{p\}$ on the well-founded side, and front(c) = $\{p\}$. In conclusion, p immediately dominates $L \setminus \{p\}$ in SFG(Π, **A**). In particular, p immediately dominates L in SFG(Π, **A**) □

For unary programs, our domination-based approach can even simulate WFD.

Theorem 3.13

Given a unary program Π and a body-saturated assignment **A**. Let $U \subseteq$ atom(Π) and $p \in U \setminus \mathbf{A^F}$. The atom p immediately dominates U in SFG(Π, **A**) if and only if ES$_\Pi(U \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{$body$(r) \mid r \in \Pi,\ p \in$ body$(r)^+\}$.

Proof. The strategy of this proof follows the one from Theorem 3.12.

The implication (⇒) holds by Theorem 3.11. It remains to show (⇐) that if ES$_\Pi(U \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{$body$(r) \mid r \in \Pi,\ p \in$ body$(r)^+\}$ then the atom p immediately dominates U in SFG(Π, **A**). Let ES$_\Pi(U \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{$body$(r) \mid r \in \Pi,\ p \in$ body$(r)^+\}$. By Lemma 3.5, ES$_\Pi$-cut$(U \setminus \{p\})$ is a support cut in SFG(Π, **A**), and by Theorem 3.6, front(ES$_\Pi$-cut$(U \setminus \{p\})) =$ ES$_\Pi(U \setminus \{p\}) \setminus \mathbf{A^F}$. Hence, by assumption that ES$_\Pi(U \setminus \{p\}) \setminus \mathbf{A^F} \subseteq \{$body$(r) \mid r \in \Pi,\ p \in$ body$(r)^+\}$, we have front(ES$_\Pi$-cut$(U \setminus \{p\})) \subseteq \{$body$(r) \mid r \in \Pi,\ p \in$ body$(r)^+\}$. Next, let ES$_\Pi$-cut$(U \setminus \{p\}) = (S, W)$. We construct a new cut c = (S', W') where $S' = S \setminus$ front(ES$_\Pi$-cut$(U \setminus \{p\}))$ and $W' = W \cup$ front(ES$_\Pi$-cut$(U \setminus \{p\}))$. Then, we have back(c) = back(ES$_\Pi$-cut$(U \setminus \{p\})) \subseteq$ body(Π) and front(c) ⊆ atom(Π). Therefore, c is an atom cut in SFG(Π, **A**).

It remains to show that front(c) = $\{p\}$. Let $r \in \Pi$ be a rule such that body$(r) \in$ front(ES$_\Pi$-cut$(U \setminus \{p\})$). By assumption, $p \in$ body$(r)^+$. As Π is unary, |body$(r)^+$| ≤ 1, we have body$(r)^+ = \{p\}$. Then, by Definition 3.1, sup-dep$_\Pi$(body$(r)) = \{p\}$.

Hence, c is an atom cut in SFG($\Pi, \mathbf{A}$) with atoms in $U \setminus \{p\}$ on the well-founded side, and front(c) = $\{p\}$. In conclusion, p immediately dominates $U \setminus \{p\}$, and in particular, p immediately dominates U in SFG($\Pi, \mathbf{A}$). □

The above Theorem relates to Janhunen's Lemma 24 Janhunen (2006). For *positive* unary programs, that are unary programs without default negation, Janhunen's Lemma 24 states that conclusions can be derived from facts following paths drawn from the body of a rule containing a true atom to its head (which in turn can be inferred as true as well). If the truth of an atom p is derived this way in an unambiguous way, i.e., no two different derivation paths exist, then WFD can be seen as a counterpart of Janhunen's Lemma 24, following the same path backwards starting from the true atom p.

3.5 Limitations

We have seen in the previous sections how computing the dominators in the support flowgraph of a (component-) unary program w.r.t. an assignment can be used to simulate the effects of WFJ (BL) and WFD (LD, respectively). Our domination-based method also works for more general classes of programs, i.e., those that are not component-unary. In this case, however, we cannot always capture all consequences of WFJ, WFD, and the related propagators. As every program can be translated into a *binary* program (Janhunen, 2000), i.e., a program with at-most two atoms in the positive body of every rule, we demonstrate the limits of our method on a binary program.

Example 3.8

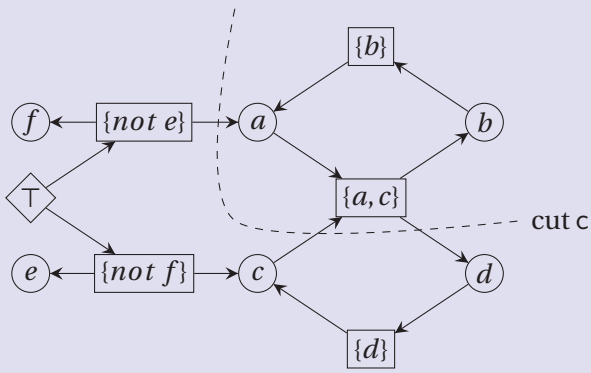
Consider the program Π with the following set of rules:

$$r_1 : a \leftarrow b \qquad r_3 : a \leftarrow not\ e \qquad r_5 : b \leftarrow a, c \qquad r_7 : c \leftarrow not\ f$$
$$r_2 : d \leftarrow a, c \qquad r_4 : e \leftarrow not\ f \qquad r_6 : f \leftarrow not\ e \qquad r_8 : c \leftarrow d$$

Verify that Π is not component-unary because $|\text{body}(r)^+ \cap \text{scc}(\text{body}(r))| = 1$ is not satisfied for $r = r_2$, $r = r_5$ respectively. Next, consider the body-saturated and unfounded-free assignment $\mathbf{A}$ given through

$$\mathbf{A} = \{\mathbf{T}a, \mathbf{T}b, \mathbf{T}c, \mathbf{T}d, \mathbf{T}\{a, c\}, \mathbf{T}\{b\}, \mathbf{T}\{d\}\}.$$

Verify that $\mathrm{ES}_{\Pi}(\{a,b\}) \setminus \mathbf{A}^{\mathbf{F}} = \{\{not\ e\}\}$. However, as can be seen from the illustration of $\mathrm{SFG}(\Pi, \mathbf{A})$ below, the body node $\{not\ e\}$ dominates neither a nor b.



In other words, every cut that separates a and b from $\{not\ e\}$ is no support cut. For instance, the cut c given in the illustration is not a support cut. Hence, a domination-based approach to BL does not conclude $\mathbf{T}\{not\ e\}$.

The reason why our method cannot always detect all consequences, e.g., of BL, becomes obvious when we recall that the support flowgraph is constructed in a way that every path to a set of atoms represents a possible derivation of their truth. Then, for programs that are not component-unary support may have to flow along multiple paths. In Example 3.8 for instance, support for the atoms in $\{a,b\}$ has to pass through the body nodes $\{not\ e\}$ and $\{not\ f\}$.

In general, however, programs can become (component-) unary as truth values are assigned during search. It is also important to note that for programs which are not (component-) unary, our domination-based method can still simulate WFJ and WFD (BL and LD, respectively) on the maximal (component-) unary sub-program.

A class of conditions that are naturally expressed in form of component-unary program is given through REACHABILITY, that we will analyse in Section 5.3. For those conditions, we can compute the consequences from unfounded set inference via dominators in the support flowgraph.

Another concern is the practical realisation of our technique, as implementing Georgiadis and Tarjan's linear-time algorithm (2004) for finding all dominators in a flowgraph represents a challenging engineering exercise. It relies on many sophisticated, non-standard data structures. Even more challanging is the implementation of an incremental version (Sreedhar et al., 1997).

3.6 Related Work

A straightforward method to computing consequences from unfounded sets is a reduction to the application of UP on SAT. This may require the introduction of additional atoms. As shown by Lifschitz and Razborov (2006), it is unlikely that, in general, a polynomial-size translation from ASP to SAT would not require additional atoms. Evidence is provided by the encoding of Lin and Zhao (2002) that has exponential space complexity. Another result, shown by Niemelä (1999), is that ASP cannot be translated into SAT in a faithful and modular way. Sacrificing modularity, however, Janhunen (2006) showed that component-wise translations are feasible. E.g., the non-modular program transformations based on level rankings (Niemelä, 2008) devised by Janhunen and Niemelä (2011) require only sub-quadratic space. Whilst their level mapping allows for computing additional information with UP, it cannot replace unfounded set inference.

The following example demonstrates that UP and WFN on the level mapping transformation of a program can hinder the propagation of unfounded sets. The gap can be closed with WFJ.

Example 3.9

Consider the program Π which is a slight variation of the one from Example 2.11, consisting of the following rules:

$$r_1 : p \leftarrow not\ q \qquad r_3 : r \leftarrow p \qquad r_5 : s \leftarrow r,\ not\ q$$
$$r_2 : q \leftarrow not\ p \qquad r_4 : r \leftarrow s$$

It has the same answer sets as the program from Example 2.11, and even its level mapping transformation LP2LP(Π) is similar to the one shown in Example 2.18. In fact, we only have to replace the fifth line with the following:

$$s \leftarrow \mathbf{bt}(r_5) \qquad \mathbf{bt}(r_5) \leftarrow r,\ not\ q$$

Now, consider the assignment $\mathbf{A}$ given through

$$\mathbf{A} = \{\mathbf{T}r, \mathbf{Fext}(s), \mathbf{Flvl}(s)_1,\ s,\ \mathbf{ext}(s), \mathbf{F}\{\mathbf{lvl}(r)_1,\ not\ r\}\}.$$

Verify that $\mathrm{UP}^*(\Delta_{\mathrm{LP2LP}(\Pi)}, \mathbf{A}) = \mathbf{A}$ and $\mathrm{WFN}^*[2^{\mathrm{atom}(\Delta_{\mathrm{LP2LP}(\Pi)})}](\Delta_{\mathrm{LP2LP}(\Pi)}, \mathbf{A}) = \mathbf{A}$. Hence, the joint fixpoint application of UP and WFN assings no further values.

However, since $\mathrm{ES}_{\{r,s\}}(\Pi) \setminus \mathbf{F} = \{\{p\}\}$, we have $\mathrm{WFJ}[\mathrm{loop}(\Pi)](\Pi, \mathbf{A}) = \mathbf{A} \cup \{\mathbf{T}\{p\}\}$, i.e., the application of BL concludes $\mathbf{T}\{p\}$.

We invite the interested reader to verify that $\mathbf{T}\{p\}$ is still not inferred by UP and WFN on the transformation of Janhunen and Niemelä that uses strong ranking constraints.

This makes a good case for unfounded set inference, and BL in particular.

An advantage of native ASP solvers like *clasp* (Gebser et al., 2007b), *dlv* (Leone et al., 2006), and *smodels* (Simons et al., 2002) over SAT-based systems (Giunchiglia et al., 2006; Janhunen and Niemelä, 2011; Lin and Zhao, 2002) is that they can integrate propagators for unfounded sets. Hence, those systems have the potential to propagate more consequences from unfounded sets using less space and time. Formal means for analysing ASP computations in terms of inference were introduced by Gebser and Schaub (2013). According which, *smodels' atmost* and *dlv's greatest unfounded set detection*, both compute WFN and FL, respectively, using linear time. Similarly, *clasp's unfounded set check* computes FL (Anger et al., 2006). Gebser and Schaub also identified the backward propagation operations for unfounded sets, i.e., WFJ and BL. A method that can be used to propagate BL has been proposed by Chen et al. (2013). It is inefficient due to high computational costs (i.e., cubic in the size of the program), but complete even for programs that are not component-unary. Their idea is to first falsify all atoms in loops without external support, and then search for a rule whose exclusion from the program would yield a loop without external support. In conclusion, the body of such rule provides the single external support for all atoms in the loop. In turn, a set of nogoods is generated to enforce the truth of the body if an atom in the loop is true. The algorithm could be extended to capture LD inference at additional costs in terms of asymptotic time complexity, e.g., by searching for an atom that, when all rules containing it in their positive part are removed from the program, yield a loop without external support. This relates to pushing all bodies from the support side of a support-cut to the well-founded side.

We have devised a linear-time approximation of WFJ and shown under which conditions our method simulates WFJ and BL, respectively. Moreover, we have put forward WFD and LD as new forms of inference that can draw additional consequences from unfounded sets. Our approach uses a reduction to the task of finding all dominators in the support flowgraph of a program, for which efficient algorithms exist. For instance, Georgiadis and Tarjan's algorithm (2004) runs in linear time, and computing all dominators can be made incremental (Sreedhar

et al., 1997).

3.7 Conclusions

This chapter has narrowed the gap to efficiently propagating the consequences from WFJ and BL. Our main contribution was a linear-time approximation of WFJ based on a reduction to finding all dominators in the support flowgraph. This gave rise to novel forms of inference, WFD and DL, which can be approximated using the same techniques. We have outlined classes of logic programs for which our approximations simulate WFJ and BL, and WFD and LD, respectively. As we shall see in Chapter 5, this includes REACHABILITY conditions that are relevant to a range of real world applications.

Despite our best efforts, efficient algorithms for fully propagating WFJ and WFD, or proof of a quadratic lower bound on their time complexity remain as open problems.

Moreover, our work raises a number of questions. The foremost one concerns an experimental comparison with alternative approaches to unfounded set inference, e.g., Janhunen and Niemelä's level mapping transformations. To our knowledge, there is no strong empirical evidence that any single approach is in practice strictly better than all others. Whilst it is left with the programmer to choose a technique that best fits a problem domain, a step towards automating this selection might be to look at some extra bookkeeping to monitor atom activity in unfounded set inference.

Another open problem is whether WFJ, or WFD, or restrictions thereof can be simulated by UP and WFN using (small) program transformations. Gebser and Schaub argue that WFJ cannot be simulated by UP and WFN, but they did not look at program transformations. We showed that our approximation of BL computes consequences from a program that cannot be inferred by UP and FL on Janhunen and Niemelä's level mapping transformation, but it is not known whether the gap can be closed by alternative encodings. This direction will not be further investigated in this thesis.

Chapter 4

Translation-based Constraint Answer Set Solving

We now describe our translational approach to constraint answer set solving. In this approach, all parts of the CASP encoding are mapped into ASP for which highly efficient, conflict-driven solvers are available.

- First, we present the foundations of the key idea. Based on Lifschitz and Turner's splitting set theorem, we show how a given constraint program can be compiled into a (normal logic) program by adding an ASP reformulation of all CP constructs that appear in the constraint program.

- We present various ASP representations of the variables' domains: the value, bound, and range encoding, and mixed variants thereof.

- Given these, we consider four different but generic ASP encodings that work for any constraint: the direct, support, bound and range encoding. Each represents constraints in a different way.

- We provide theoretical results on their propagation strength, i.e., what type of local consistency is achieved by the UP inference of any ASP solver.

Because atoms will be shared between the ASP encodins of the constraints, modern solving techniques like conflict-driven nogood learning (CDNL; Gebser et al., 2007a) can exploit interdependencies between constraint and use this information to improve propagation.

4.1 Foundations

To begin, recall that the problem of computing constraint answer sets of a constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ can be broken down into two tasks:

1. computing configurations for the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$, i.e., pairs $(A, \mathrm{sat}_{\mathsf{C}}(A))$, and

2. computing the answer sets of $\Pi(\mathrm{atom}(\mathrm{sat}_{\mathsf{C}}(A)))$, i.e., the program Π with externals over $\mathscr{C}$.

Whilst hybrid approaches to constraint answer set solving employ a CP solver for the former, and an ASP solver for the latter, a translation-based approach reduces the problem of computing configurations for the CSP to the one of computing answer sets of a program Ψ with externals from $\mathscr{C}$ (and possibly further auxiliary atoms).

Definition 4.1: ASP Encoding of a Constraint Satisfaction Problem

Let $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ be a constraint program where Π is a program with externals over constraint atoms $\mathscr{C}$. An *ASP encoding* of the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$ is a program Ψ over an alphabet $\mathscr{E} \supseteq \mathscr{C}$ such that

- the answer sets of Ψ correspond one-to-one to the configurations of $(\mathsf{V}, \mathsf{D}, \mathsf{C})$, in particular, there is a bijective function $\mathrm{assign}_{\mathsf{V},\mathsf{D}}(X)$ that maps each answer set X of Ψ into the corresponding assignment A, and

- for each answer set X of Ψ and constraint atom $c \in \mathscr{C}$ holds that $c \in X$ if and only if $\mathrm{constraint}(c) \in \mathrm{sat}_{\mathsf{C}}(A)$.

Extracting a configuration $(A, \mathrm{sat}_{\mathsf{C}}(A))$ of the CSP from an answer set of Ψ is easy: If X is an answer set of Ψ then the set $\mathrm{sat}_{\mathsf{C}}(A)$ is represented by the constraint atoms $X \cap \mathscr{C}$. The extraction of the corresponding assignment A, however, depends on the mapping $\mathrm{assign}_{\mathsf{V},\mathsf{D}}$, i.e., on how the variables' domains are represented in Ψ. In this chapter, we will consider three different representations that encode values, bounds, and ranges in the domain of a variable. As we shall see, the assignment $\mathsf{v} = i$ can be represented by atom $[\![\mathsf{v} = i]\!] \in X$ in our value encoding, by atom $[\![\mathsf{v} \in [i, i]]\!] \in X$ in our range encoding, and by atoms $[\![\mathsf{v} \leq i]\!] \in X$ and $[\![\mathsf{v} \leq i-1]\!] \notin X$ in our bounds encoding, where $i-1$ is the predecessor of the value i in the domain of v.

Given this, we can compute the constraints answer sets of the constraint program $\mathbb{P}$ by encoding the corresponding CSP into an ASP encoding Ψ and comput-

ing the answer sets of the (joint) program $\Pi \cup \Psi$.

Theorem 4.1

Let $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ be a constraint program where Π is a program with externals over constraint atoms $\mathscr{C}$, and the program Ψ over $\mathscr{E} \supseteq \mathscr{C}$ be an ASP encoding of the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$. The set $X \subseteq \text{atom}(\Pi) \cup \mathscr{E}$ is an answer set of $\Pi \cup \Psi$ if and only if $(X \setminus \mathscr{E}, \text{assign}_{\mathsf{V},\mathsf{D}}(X \cap \mathscr{E}))$ is a constraint answer sets of $\mathbb{P}$.

Proof. The proof follows from the splitting set theorem (Lifschitz and Turner, 1994) that we introduced in the Background section of this thesis. We show both implications of the proposition.

($\Rightarrow$) Let $X \subseteq \text{atom}(\Pi) \cup \mathscr{E}$ be an answer set of $\Pi \cup \Psi$. By construction, $\mathscr{E}$ is a splitting set for $\Pi \cup \Psi$. Hence, the splitting set theorem applies and we get $X \cap \mathscr{E}$ is an answer set of Ψ and $X \setminus \mathscr{E}$ is an answer set of $\Pi(X \cap \mathscr{E})$. Then, by Definition 4.1, $X \cap \mathscr{E}$ corresponds to a configuration $(A, \text{sat}_{\mathsf{C}}(A))$ of the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$ and $X \cap \mathscr{E} = \text{atom}(\text{sat}_{\mathsf{C}}(A)) \cap \text{atom}(\Pi)$, where $A = \text{assign}_{\mathsf{V},\mathsf{D}}(X \cap \mathscr{E})$. In conclusion, $(X \setminus \mathscr{E}, A)$ is a constraint answer set of $\mathbb{P}$.

($\Leftarrow$) Let (X, A) be a constraint answer set of $\mathbb{P}$. Then, by definition, X is an answer set of $\Pi(\text{atom}(\text{sat}_{\mathsf{C}}(A)) \cap \text{atom}(\Pi))$ and $(A, \text{sat}_{\mathsf{C}}(A))$ is a configuration to the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$. Then, by Definition 4.1, $(A, \text{sat}_{\mathsf{C}}(A))$ corresponds to an answer set Y of Ψ such that $c \in Y$ if and only if $\text{constraint}(c) \in \text{sat}_{\mathsf{C}}(A)$ for each $c \in \mathscr{C}$. Hence, $Y \cap \mathscr{C} = \text{atom}(\Pi) \cap \text{atom}(\text{sat}_{\mathsf{C}}(A))$. Since $\mathscr{E} \supseteq \mathscr{C}$ is a splitting set for $\Pi \cup \Psi$, separating $\Pi \cup \Psi$ into Π and Ψ by construction, the splitting set theorem applies and we get $X \cup Y$ is an answer set of $\Pi \cup \Psi$. □

Whilst Lifschitz and Turner's splitting set theorem facilitates the reduction of the problem of computing the constraints answer sets of a constraint program to computing the answer sets of a single, joint ASP encoding, it also provides us with the foundations of a modular design principle for constructing ASP encodings in an automated fashion, and lays the foundations to dissecting that encoding in order to allow a study of the effect of ASP inference on sub-programs, i.e., encodings of individual constraints independently from each other. For instance, the ASP encoding Ψ of a CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$ can be constructed from the union of

- a program Ψ_{V} encoding of the variables' domains, and
- a program Ψ_c with externals over $\text{atom}(\Psi_{\mathsf{V}})$ for each constraint $c \in \mathsf{C}$, representing the conditions on the variables' domains induced by c.

The construction can be implemented as preprocessing, so that existing ASP systems can be applied to compute constraint answer sets without changing their source code. This allows for programmers to select the solver that best fit their needs. We will use conflict-driven solvers to enable conflict-driven constraint answer set solving. This has a range of advantages over hybrid CASP and CP systems, including the following:

- All constraints are made transparent, there is no black-box propagation algorithm.
- Hence, no scheduling of constraints is required. As atoms will be shared between constraints, and all constraints are always propagated at the same time by the inference of the underlying ASP solver.
- CDNL can exploit constraint interdependencies. This can improve propagation between constraints and contribute to search heuristics.
- Advanced dynamic selection heuristics from conflict-driven solvers can be used. There is no need for programming search.

The remainder of this and the following chapter concern the two main challenging tasks of encoding the variables' domains and constraints into ASP, that are, the programs Ψ_V and Ψ_c above, and investigating the effect of ASP inference on $\Psi_V \cup \Psi_c$ in terms of local consistency on the original constraint.

We start our study with four generic encodings: the value, support, bound, and range encoding. Each represents constraints in a different, generic way. We will demonstrate that the UP inference of any ASP solver on these encodings provides a propagator of the original constraint that can hinder propagation, or achieve arc, bound, and range consistency, respectively.

We will make an assumption on the structure of the domains, though, to save the reader from multiple superscripts and simplify the presentation of encodings, theorems, and their proofs.

Assumption: Variables' Domains

The domain of a variable is the interval of integer values $[1, d]$.

The encoding of a consistent representation of the variables' domains will be shared between the ASP encodings of all constraints. Hence, it has to be chosen carefully, considering the options for encoding individual constraints at hand. Since

constraint encodings are programs with externals over atoms representing the domains of the variables in their scope, they are independent of the actual implementation of the variables' domains representation. This allows for encodings that mix different representations.

4.2 Value Encoding

Our first encoding is the popular and straightforward *value encoding*, similar to the *direct encoding* by Walsh (2000). In the value encoding, a new atom is introduced for each variable $\mathsf{v} \in \mathsf{V}$ and each value i from the variable's domain. We will denote this atom $[\![\mathsf{v} = i]\!]$. An assignment to $[\![\mathsf{v} = i]\!]$ will indicate whether the value i can be assigned to v. Hence, $[\![\mathsf{v} = i]\!]$ will be assigned true if v takes the value i, and false if the value i has been removed from the domain of v (or, rather, v takes a value different from i).

Definition 4.2: Value Encoding of Variables' Domains

Our value encoding of the variables' domains, denoted ASP-VALUE[V], is specified as follows, where $\mathsf{v} \in \mathsf{V}$.

$$\{[\![\mathsf{v} = 1]\!], \ldots, [\![\mathsf{v} = d]\!]\} \leftarrow \tag{4.1}$$

$$\leftarrow not\ [\![\mathsf{v} = 1]\!], \ldots, not\ [\![\mathsf{v} = d]\!] \tag{4.2}$$

$$\leftarrow 2\ \{[\![\mathsf{v} = 1]\!], \ldots, [\![\mathsf{v} = d]\!]\} \tag{4.3}$$

The choice rule (4.1) encodes all possible assignments for a variable v, while the remaining rules encode a consistent assignment, i.e., that there must be at least one value for each v, and that v takes at-most one value. We specify that v takes at least one value by the integrity constraint (4.2). Cardinality constraint rules (4.3) express that v cannot take two values, and hence, v takes at-most one value.

Overall, the value encoding introduces $\mathcal{O}(nd)$ new atoms and $\mathcal{O}(n)$ rules, where $n = |\mathsf{V}|$ and d is the largest domain size.

Example 4.1

Consider the variable v with $\mathrm{dom}(\mathsf{v}) = [1,3]$. The value encoding of v is

$$\{[\![\mathsf{v} = 1]\!], [\![\mathsf{v} = 2]\!], [\![\mathsf{v} = 3]\!]\} \leftarrow$$

$$\leftarrow not\ [\![\mathsf{v} = 1]\!], not\ [\![\mathsf{v} = 2]\!], not\ [\![\mathsf{v} = 3]\!]$$

$$\leftarrow 2\ \{[\![\mathsf{v} = 1]\!], [\![\mathsf{v} = 2]\!], [\![\mathsf{v} = 3]\!]\}$$

which can be compiled into a program without choice and cardinality constraint rules (see Section 2.5). From this, we can construct the set of nogoods $\Delta_{\text{ASP-VALUE}[\{\mathsf{v}\}]}$. The situation where no value has been removed from v's domain is represented in an assignment $\mathbf{A}$ such that $[\![\mathsf{v} = i]\!] \notin \mathbf{A}^{\mathbf{T}} \cup \mathbf{A}^{\mathbf{F}}$ for $i \in [1,3]$. The removal of a value i from v's domain is indicated by $\mathbf{F}[\![\mathsf{v} = i]\!] \in \mathbf{A}^{\mathbf{F}}$, for instance, $\mathbf{F}[\![\mathsf{v} = 2]\!] \in \mathbf{A}$ if 2 was removed. A situation where only a single value j is left in v's domain, or v is assigned j, is represented by $\mathbf{T}[\![\mathsf{v} = j]\!] \in \mathbf{A}$ and $\mathbf{F}[\![\mathsf{v} = i]\!] \in \mathbf{A}$ for $i \neq j$. Some examples are given in the table below.

1	2	3	Property of $\mathbf{A}$
✓	✓	✓	$[\![\mathsf{v} = 1]\!], [\![\mathsf{v} = 2]\!], [\![\mathsf{v} = 3]\!] \notin \mathbf{A}^{\mathbf{T}} \cup \mathbf{A}^{\mathbf{F}}$
✓		✓	$\mathbf{F}[\![\mathsf{v} = 2]\!] \in \mathbf{A}, [\![\mathsf{v} = 1]\!], [\![\mathsf{v} = 3]\!] \notin \mathbf{A}^{\mathbf{T}} \cup \mathbf{A}^{\mathbf{F}}$
		✓	$\mathbf{F}[\![\mathsf{v} = 1]\!], \mathbf{F}[\![\mathsf{v} = 2]\!], \mathbf{T}[\![\mathsf{v} = 3]\!] \in \mathbf{A}$

The table identifies values that have not been removed from the domain of v (✓) and properties that can be observed in $\mathbf{A}$.

The expert reader will notice that there is a slightly more compact representation of the conditions (4.1–4.3) if choice rules with bounds are allowed. As those are typically compiled into above rules, we will refrain from introducing additional syntactic sugar.

We proceed with representing constraints with ASP. Observe that every constraint can be encoded into a program with externals over the above representation of the domains of the variables in its scope. Let $c \in \mathsf{C}$ and $\text{scope}(c) = (\mathsf{v}_1, \ldots, \mathsf{v}_n)$. In the value encoding, c is decomposed into forbidden combination of values. Let $\mathsf{v}_1 = i_1, \ldots, \mathsf{v}_n = i_n$ be such a forbidden combination, we post a rule of the form

$$[\![\overline{c}]\!] \leftarrow [\![\mathsf{v}_1 = i_1]\!], \ldots, [\![\mathsf{v}_n = i_n]\!] \tag{4.4}$$

and repeat the process for each forbidden combination. A rule of the form (4.4) can be read as whenever $\mathsf{v}_1 = i_1, \ldots, \mathsf{v}_n = i_n$ holds, i.e., a forbidden combination is satisfied, then the constraint is violated, represented by the atom $[\![\overline{c}]\!]$. We use default negation to access information on whether a constraint $c \in \mathsf{C}$ is satisfied rather than violated, expressed by the atom $[\![c]\!]$. To ensure consistency, i.e., either $[\![c]\!]$ or $[\![\overline{c}]\!]$ is contained in any solution, we post the following rule:

$$[\![c]\!] \leftarrow not\ [\![\overline{c}]\!] \tag{4.5}$$

When a relation is represented by allowed combinations of values, however, then all forbidden combinations have to be deduced. Alternatively, we can encode rules

for $[[c]]$, encoding all allowed combination of values. Let now $\mathsf{v}_1 = i_1, \ldots, \mathsf{v}_n = i_n$ be an allowed combination, we post

$$[[c]] \leftarrow [[\mathsf{v}_1 = i_1]], \ldots, [[\mathsf{v}_n = i_n]]$$

and repeat the process for each allowed combination. Now we can express $[[\overline{c}]]$ via default negation of $[[c]]$

$$[[\overline{c}]] \leftarrow not\ [[c]]$$

In the remainder of this thesis, however, the process of decomposing a constraint into ASP will involve the introduction of rules with head $[[\overline{c}]]$. It is understood that a rule of the form (4.5) has to be defined whenever the atom $[[c]]$ is required. For the sake of brevity, we will omit making it explicit in the sequel.

Example 4.2

Consider two variables v_1 and v_2 with $\mathrm{dom}(\mathsf{v}_1) = [2,4]$ and $\mathrm{dom}(\mathsf{v}_2) = [1,3]$, and a constraint c where $\mathrm{scope}(c) = (\mathsf{v}_1, \mathsf{v}_2)$ and $\mathrm{range}(c) = \{(i, j) \mid i \in \mathrm{dom}(\mathsf{v}_1), j \in \mathrm{dom}(\mathsf{v}_2), i \leq j\}$. In other words, c is the constraint $\mathsf{v}_1 \leq \mathsf{v}_2$. The value encoding of c, using forbidden combinations of values, is given through the following set of rules.

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 2]],\ [[\mathsf{v}_2 = 1]] \qquad [[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 4]],\ [[\mathsf{v}_2 = 1]]$$

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 3]],\ [[\mathsf{v}_2 = 1]] \qquad [[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 4]],\ [[\mathsf{v}_2 = 2]]$$

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 3]],\ [[\mathsf{v}_2 = 2]] \qquad [[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 4]],\ [[\mathsf{v}_2 = 3]]$$

Unfortunately, the value encoding hinders propagation:

Theorem 4.2: Unit Propagation on the Value Encoding

Enforcing arc consistency on the binary decomposition of a constraint prunes more values from the variables' domains than UP on its value encoding.

Proof. A SAT-equivalent of the theorem was shown by Walsh (2000), by considering a constraint on two variables over two values that rules out every possible combination of values. For instance, consider the constraint c with $\mathrm{scope}(c) = (\mathsf{v}_1, \mathsf{v}_2)$ and $\mathrm{range}(c) = \emptyset$, where $\mathrm{dom}(\mathsf{v}_1) = \mathrm{dom}(\mathsf{v}_2) = [1,2]$. A value encoding of c is the

program Ψ_c given by the set of rules

$$
\begin{aligned}
r_{1,1}: &\quad [[\overline{c}]] \leftarrow [[\mathsf{c}_1 = 1]],\ [[\mathsf{c}_2 = 1]] \\
r_{1,2}: &\quad [[\overline{c}]] \leftarrow [[\mathsf{c}_1 = 1]],\ [[\mathsf{c}_2 = 2]] \\
r_{2,1}: &\quad [[\overline{c}]] \leftarrow [[\mathsf{c}_1 = 2]],\ [[\mathsf{c}_2 = 1]] \\
r_{2,2}: &\quad [[\overline{c}]] \leftarrow [[\mathsf{c}_1 = 2]],\ [[\mathsf{c}_2 = 2]]
\end{aligned}
$$

representing the following set of nogoods:

$$
\begin{aligned}
&\{\{\mathbf{T}[[\overline{c}]], \mathbf{F}\,\mathrm{body}(r_{1,1}), \mathbf{F}\,\mathrm{body}(r_{1,2}), \mathbf{F}\,\mathrm{body}(r_{2,1}), \mathbf{F}\,\mathrm{body}(r_{2,2})\}\} \\
\cup &\{\{\mathbf{F}[[\overline{c}]], \mathbf{T}\,\mathrm{body}(r_{i,j})\} \mid i, j \in \{1,2\}\} \\
\cup &\{\{\mathbf{F}\,\mathrm{body}(r_{i,j}), \mathbf{T}[[\mathsf{c}_1 = i]], \mathbf{T}[[\mathsf{c}_2 = j]]\} \mid i, j \in \{1,2\}\} \\
\cup &\{\{\mathbf{T}\,\mathrm{body}(r_{i,j}), \mathbf{F}[[\mathsf{c}_1 = i]]\}, \{\mathbf{T}\,\mathrm{body}(r_{i,j}), \mathbf{F}[[\mathsf{c}_2 = j]]\} \mid i, j \in \{1,2\}\} \\
&\subseteq \Delta_{\Psi_c \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \mathsf{v}_2\}]}
\end{aligned}
$$

The set $\Delta_{\Psi_c \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \mathsf{v}_2\}]}$ does not contain any unit nogood. In particular

$$
\mathrm{UP}(\Delta_{\Psi_c \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \mathsf{v}_2\}]}, \{\mathbf{F}[[\overline{c}]]\}) = \{\mathbf{F}[[\overline{c}]]\}
$$

i.e., UP does not prune any value. On the other hand, enforcing arc consistency on this binary constraint removes all values from the variables' domains. □

Observe that the programs from the above proof, Example 4.2, and the value encoding of any other constraint are tight. ASP inference that aims at unfounded sets which are loops of the program, e.g., FL, BL, or LD, has therefore no effect, and cannot complement UP in this setting.

4.3 Support Encoding

The *support encoding* has been proposed by Gent (2002) in the domain of SAT to tackle the limitations of value encodings. We transfer here Gent's idea to ASP, i.e., we encode support information for assignments rather than conflicts.

Support is defined for a possible assignment to a variable, and is given by the set of values for another variable which allow this assignment. Formally, a *binary support* for a variable v to take the value i across a constraint $c \in \mathsf{C}$ is the set of values $\{j_1, \ldots, j_m\} \subseteq \mathrm{dom}(\mathsf{v}')$ of another variable in $\mathsf{v}' \in \mathrm{scope}(c) \setminus \{\mathsf{v}\}$ that do not violate the constraint if $\mathsf{v} = i$. Whilst binary support is a special case of support, observe that, binary support equals to the more general notion if the constraint is binary.

The support encoding of a constraint works on the value representation of the domains of the variables in its scope. Let $c \in \mathsf{C}$ and $\text{scope}(c) = (\mathsf{v}_1, \ldots, \mathsf{v}_n)$. We encode binary support by a rule of the form

$$[[\overline{c}]] \leftarrow [[\mathsf{v} = i]], not\ [[\mathsf{v}' = j_1]], \ldots, not\ [[\mathsf{v}' = j_m]] \tag{4.6}$$

where the set the values $\{j_1, \ldots, j_m\}$ for variable v' is the support for $\mathsf{v} = i$ across c. A rule of the form (4.6) can be read as whenever $\mathsf{v} = i$, then at least one of its supports must hold, otherwise the constraint is violated. In the support encoding of a constraint $c \in \mathsf{C}$, we encode binary support for each pair of distinct variables $\mathsf{v}, \mathsf{v}' \in \text{scope}(c)$, and for each value $i \in \text{dom}(\mathsf{v})$.

Example 4.3

Reconsider the setting from Example 4.2, i.e., two variables v_1 and v_2 with the domains $\text{dom}(\mathsf{v}_1) = [2,4]$ and $\text{dom}(\mathsf{v}_2) = [1,3]$, and the constraint $c = \mathsf{v}_1 \leq \mathsf{v}_2$. The support encoding of c is given through the following set of rules.

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 1]],\ not\ [[\mathsf{v}_2 = 1]],\ not\ [[\mathsf{v}_2 = 2]],\ not\ [[\mathsf{v}_2 = 3]],\ not\ [[\mathsf{v}_2 = 4]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 2]],\ not\ [[\mathsf{v}_2 = 2]],\ not\ [[\mathsf{v}_2 = 3]],\ not\ [[\mathsf{v}_2 = 4]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 3]],\ not\ [[\mathsf{v}_2 = 3]],\ not\ [[\mathsf{v}_2 = 4]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 = 4]],\ not\ [[\mathsf{v}_2 = 4]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_2 = 1]],\ not\ [[\mathsf{v}_1 = 1]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_2 = 2]],\ not\ [[\mathsf{v}_1 = 1]],\ not\ [[\mathsf{v}_1 = 2]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_2 = 3]],\ not\ [[\mathsf{v}_1 = 1]],\ not\ [[\mathsf{v}_1 = 2]],\ not\ [[\mathsf{v}_1 = 3]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_2 = 4]],\ not\ [[\mathsf{v}_1 = 1]],\ not\ [[\mathsf{v}_1 = 2]],\ not\ [[\mathsf{v}_1 = 3]],\ not\ [[\mathsf{v}_1 = 4]]$$

Verify that UP prunes the value 4 from v_1's domain and the value 1 from v_2's domain, providing an advantage over the value encoding where UP does not prune any value.

In general, UP on the support encoding achieves arc consistency.

Theorem 4.3: Unit Propagation on the Support Encoding

UP on the support encoding enforces arc consistency on the binary decomposition of the original constraint.

Proof. A SAT-equivalent of the Theorem was shown by Gent (2002).

Let c be any constraint and $\mathbf{A}$ be an assignment to the atoms in a value representation of the variables in the scope of the constraint, i.e., the atoms in the pro-

gram ASP-VALUE[scope(c)], representing the current set of values in the domain of the variables. Suppose UP has been run to completion, extending $\mathbf{A} \cup \{\mathbf{F}[[\overline{c}]]\}$ (i.e., the constraint c shall not be violated) to the conflict-free assignment $\mathbf{A}'$. The key argument is that all nogoods represented by the support encoding of c contain at most one positive literal of an unassigned atom, i.e., all nogoods are Horn-style and can be solved in linear time via UP. Since the support encoding will prune any values which are not arc consistent (cf. Gent, 2002), the assignment $\mathbf{A}'$ is arc consistent. □

4.4 Bound Encoding

A different perspective on the domains of variables is provided by our *bound encoding*. Similar to the *order encoding* of Tamura et al. (2006), a new atom is introduced for each variable v and value i to represent whether the value of v is bounded from above by i, i.e., $\mathsf{v} \leq i$. We will denote this atom $[[\mathsf{v} \leq i]]$. An assignment to $[[\mathsf{v} \leq i]]$ will indicate whether a value less or equal to i can be assigned to v. To clarify, $[[\mathsf{v} \leq i]]$ will be assigned true if v takes the value i or any value less than i, and false if every value that is less or equal i has been removed from the domain of v (or, rather, v takes a value greater than i).

Definition 4.3: Bound Encoding of Variables' Domains

Our bound encoding of the variables' domains, denoted ASP-BOUND[V], is specified as follows, where $\mathsf{v} \in \mathsf{V}$ and $1 \leq i < d$.

$$\{[[\mathsf{v} \leq 1]], \ldots, [[\mathsf{v} \leq d]]\} \leftarrow \tag{4.7}$$

$$\leftarrow [[\mathsf{v} \leq i]], not\ [[\mathsf{v} \leq i+1]] \tag{4.8}$$

$$\leftarrow not\ [[\mathsf{v} \leq d]] \tag{4.9}$$

Similar to the value encoding, for each variable v, possible assignments are encoded by a choice rule of the form (4.7). We can encode the *channelling condition* that $\mathsf{v} \leq i$ implies $\mathsf{v} \leq i+1$ by consistency constraints of the form (4.8), for $1 \leq i < d$, in order to ensure a consistent set of bounds. To guarantee non-empty domains, i.e., a value must be assigned to each variable v, the integrity constraints of the form (4.9) specifies that $\mathsf{v} \leq d$ must hold.

Overall, the bound encoding introduces $\mathcal{O}(nd)$ new atoms and $\mathcal{O}(nd)$ rules, where $n = |\mathsf{V}|$ and d is the largest domain size.

Example 4.4

Consider the variable v with dom(v) = [1,4]. The bound encoding for v is

$$\{[[\mathsf{v} \leq 1]], [[\mathsf{v} \leq 2]], [[\mathsf{v} \leq 3]], [[\mathsf{v} \leq 4]]\} \leftarrow$$

$$\leftarrow [[\mathsf{v} \leq 1]], not\ [[\mathsf{v} \leq 2]]$$

$$\leftarrow [[\mathsf{v} \leq 2]], not\ [[\mathsf{v} \leq 3]]$$

$$\leftarrow [[\mathsf{v} \leq 3]], not\ [[\mathsf{v} \leq 4]]$$

$$\leftarrow not\ [[\mathsf{v} \leq 4]]$$

From this, we can construct the set of nogoods $\Delta_{\text{ASP-BOUND}[\{\mathsf{v}\}]}$. The situation where no value has been removed from v's domain is represented by an assignment $\mathbf{A}$ such that $[[\mathsf{v} \leq i]] \notin \mathbf{A^T} \cup \mathbf{A^F}$ for $i \in [1,3]$. An updated lower bound i of v's domain is indicated by $\mathbf{F}[[\mathsf{v} \leq i-1]] \in \mathbf{A}$, for instance, $\mathbf{F}[[\mathsf{v} \leq 2]] \in \mathbf{A}$ if the new lower bound is 3, and the channelling conditions also ensure $\mathbf{F}[[\mathsf{v} \leq 1]] \in \mathbf{A}$. On the other hand, an updated upper bound i of v's domain is indicated by $\mathbf{T}[[\mathsf{v} \leq i]] \in \mathbf{A}$, for instance, $\mathbf{T}[[\mathsf{v} \leq 3]] \in \mathbf{A}$ if the new upper bound is 3. The channelling conditions also ensure $\mathbf{T}[[\mathsf{v} \leq j]] \in \mathbf{A}$ where $j > i$. The value of v is fixed to i when $\mathbf{F}[[\mathsf{v} \leq i-1]] \in \mathbf{A}$ and $\mathbf{T}[[\mathsf{v} \leq i]] \in \mathbf{A}$. Some examples are given in the table below.

1	2	3	4	Property of $\mathbf{A}$
✓	✓	✓	✓	$[[\mathsf{v} \leq 1]], [[\mathsf{v} \leq 2]], [[\mathsf{v} \leq 3]] \notin \mathbf{A^T} \cup \mathbf{A^F}, \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$
✓			✓	$[[\mathsf{v} \leq 1]], [[\mathsf{v} \leq 2]], [[\mathsf{v} \leq 3]] \notin \mathbf{A^T} \cup \mathbf{A^F}, \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$
		✓	✓	$\mathbf{F}[[\mathsf{v} \leq 1]], \mathbf{F}[[\mathsf{v} \leq 2]] \in \mathbf{A}, [[\mathsf{v} \leq 3]] \notin \mathbf{A^T} \cup \mathbf{A^F}, \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$
		✓		$\mathbf{F}[[\mathsf{v} \leq 1]], \mathbf{F}[[\mathsf{v} \leq 2]], \mathbf{T}[[\mathsf{v} \leq 3]], \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$

The table identifies values that have not been removed from the domain of v (✓) and some properties that can be observed in $\mathbf{A}$ (no complete list). Note that holes in the domain are not reflected in $\mathbf{A}$ (first vs second column).

In the bound encoding, constraints are decomposed into combinations of primitive constraints representing *conflict regions*, that are sets of bounds on the domains that contain no solution of the constraint. When the combination $l_1 \leq \mathsf{v}_1 \leq u_1, \ldots, l_n \leq \mathsf{v}_n \leq u_n$ violates the constraint $c \in \mathsf{C}$, we encode a rule of the form

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 \leq u_1]], \ldots, [[\mathsf{v}_n \leq u_n]], not\ [[\mathsf{v}_1 \leq l_1 - 1]], \ldots, not\ [[\mathsf{v}_n \leq l_n - 1]] \quad (4.10)$$

and repeat the process for each conflict region. A rule of the form (4.10) can be read as whenever $l_1 \leq \mathsf{v}_1 \leq u_1, \ldots, l_n \leq \mathsf{v}_n \leq u_n$ holds, i.e., a conflict region is satisfied, then the constraint is violated.

Example 4.5

Reconsider the setting from Example 4.2, i.e., two variables v_1 and v_2 with the domains $\mathrm{dom}(\mathsf{v}_1) = [2,4]$ and $\mathrm{dom}(\mathsf{v}_2) = [1,3]$, and the constraint $c = \mathsf{v}_1 \leq \mathsf{v}_2$. The support encoding of c is given through the following set of rules.

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_2 \leq 1]],\ not\ [[\mathsf{v}_1 \leq 1]]$$

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_2 \leq 2]],\ not\ [[\mathsf{v}_1 \leq 2]]$$

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_2 \leq 3]],\ not\ [[\mathsf{v}_1 \leq 3]]$$

Verify that UP prunes the value 4 from v_1's domain and the value 1 from v_2's domain, and hence, achieves a bound consistent assignment.

In general, UP on the bound encoding achieves bound consistency.

Theorem 4.4: Unit Propagation on the Bound Encoding

UP on the bound encoding enforces bound consistency on the original constraint.

Proof. Let c be any constraint and $\mathbf{A}$ be an assignment to the atoms in a bound representation of the variables in the scope of the constraint, i.e., the atoms in the program ASP-BOUND[scope(c)], representing the current set of bounds on the domain of the variables. Suppose UP has been run to completion, extending $\mathbf{A} \cup \{\mathbf{F}[[\overline{c}]]\}$ (i.e., the constraint c shall not be violated) to the conflict-free assignment $\mathbf{A}'$. We show by proof of contradiction that the domains are bound consistent.

Suppose there is a variable $\mathsf{v}_i \in \mathrm{scope}(c)$ such that if v_i is assigned its minimum value l_i or its maximum value u_i, then there are no compatible values of the other variables $\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_n \in \mathrm{scope}(c)$ between their respective minimum $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n$ and their maximum values $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n$, i.e., the domains are not bound consistent.

First, we analyse the case $\mathsf{v}_i = u_i$, i.e., all assignments such that $\mathsf{v}_i = u_i$ are in a conflict region $l'_1 \leq l_1 \leq \mathsf{v}_1 \leq u_1 \leq u'_1, \ldots, l'_n \leq l_n \leq \mathsf{v}_n \leq u_n \leq u'_n$ encoded by a rule r of the form (4.10). To begin with, we have $[[\mathsf{v}_j \leq l_j - 1]] \in (\mathbf{A}')^{\mathbf{F}}$ and $[[\mathsf{v}_j \leq u_j]] \in (\mathbf{A}')^{\mathbf{T}}$, representing $l_j \leq \mathsf{v}_j$ and $\mathsf{v}_j \leq u_j$ for each $1 \leq j \leq n$. Since $\mathbf{A}'$ is a conflict-free, the nogoods represented by the channelling condition (4.8), most importantly $\{\mathbf{F}[[\mathsf{v}_j \leq l_j - 1]], \mathbf{T}[[\mathsf{v}_j \leq l_j - 2]]\}$ and $\{\mathbf{T}[[\mathsf{v}_j \leq u_j]], \mathbf{F}[[\mathsf{v}_j \leq u_j + 1]]\}$ guar-

antee $[\![\mathsf{v}_j \leq l_j - 2]\!] \in (\mathbf{A}')^{\mathbf{F}}$ and $[\![\mathsf{v}_j \leq u_j + 1]\!] \in (\mathbf{A}')^{\mathbf{F}}$. Repeatedly applying this argument eventually gets us $[\![\mathsf{v}_j \leq l'_j - 1]\!] \in (\mathbf{A}')^{\mathbf{F}}$ as well as $[\![\mathsf{v}_j \leq u'_j]\!] \in (\mathbf{A}')^{\mathbf{T}}$. But then the nogood $\{\mathbf{F}\,\mathrm{body}(r), \mathbf{F}[\![\mathsf{v}_1 \leq l'_1 - 1)]\!], \mathbf{T}[\![\mathsf{v}_1, \leq u'_1]\!], \ldots, \mathbf{F}[\![\mathsf{v}_n \leq l'_n - 1]\!], \mathbf{T}[\![\mathsf{v}_n \leq u'_n]\!]\}$, represented by r (defined above), guarantees $[\![\mathsf{v}_i \leq u'_i]\!] \in (\mathbf{A}')^{\mathbf{F}}$. Since $u_i < u'_i$ and, again, by the nogoods represented through the channelling condition (4.8), we have $[\![\mathsf{v}_i \leq u_i]\!] \in (\mathbf{A}')^{\mathbf{F}}$, indicating that u_i is not in the domain of v_i. This contradicts our assumption.

The second case, where v_i is assigned its minimum value l_i, is symmetric.

Hence, we conclude that the domains are bound consistent as required. Since at least one value must be in each domain, we have a set of non-empty domains which are bound consistent. □

4.5 Range Encoding

As a third generic encoding for variables and their domains we present the *range encoding*. Similar to some work decomposing CSP (cf. Bessière et al., 2009a), in the range encoding, we represent that a variable $\mathsf{v} \in \mathsf{V}$ can take values from a discrete interval $[l, u]$, i.e., a value between l and u (inclusive). For each variable v and interval $[l, u] \subseteq [1, d] = \mathrm{dom}(\mathsf{v})$ a new atom is introduced to represent whether the value of v is in the interval $[l, u]$., i.e., $l \leq \mathsf{v} \leq u$. We denote this atom $[\![\mathsf{v} \in [l, u]]\!]$. An assignment to $[\![\mathsf{v} \in [l, u]]\!]$ will indicate whether a value in the range $[1, d]$ can be assigned to v. For this purpose, $[\![\mathsf{v} \in [l, u]]\!]$ will be assigned true if v takes any value between l and u, and false if every value in the interval $[l, u]$ has been removed from the domain of v (or, rather, v takes a value outside of the interval $[l, u]$).

Definition 4.4: Range Encoding of Variables' Domains

Our range encoding of the variables' V domains, denoted ASP-RANGE$[\mathsf{V}]$, is specified as follows, where $\mathsf{v} \in \mathsf{V}$ and $1 \leq l \leq u \leq d$.

$$[\![\mathsf{v} \in [l, u]]\!] \leftarrow not\ [\![\mathsf{v} \in [1, l-1]]\!], not\ [\![\mathsf{v} \in [u+1, d]]\!] \tag{4.11}$$

$$\leftarrow [\![\mathsf{v} \in [l+1, u]]\!], not\ [\![\mathsf{v} \in [l, u]]\!] \tag{4.12}$$

$$\leftarrow [\![\mathsf{v} \in [l, u-1]]\!], not\ [\![\mathsf{v} \in [l, u]]\!] \tag{4.13}$$

For each variable $\mathsf{v} \in \mathsf{V}$, we encode possible assignments by rules of the form (4.11) where $1 \leq l \leq u \leq d$, that conclude $\mathsf{v} \in [l, u]$ whenever there is no evidence for $\mathsf{v} \in [1, l-1]$ or $\mathsf{v} \in [u+1, d]$ via default negation. In order to ensure a consistent

set of ranges, we also encode *channelling conditions*, that are $\mathsf{v} \in [l, u]$ implies $\mathsf{v} \in [l-1, u]$ and $\mathsf{v} \in [l, u+1]$, by integrity constraints of the form (4.12) and (4.13), respectively. In other words, the channelling conditions guarantee that whenever the range $[l, u]$ contains a value from the domain of v, so does any range that includes $[l, u]$.

Overall, the range encoding introduces $\mathcal{O}(nd^2)$ new atoms and $\mathcal{O}(nd^2)$ rules, where $n = |\mathsf{V}|$ and d is the largest domain size.

Example 4.6

Reconsider the variable v with $\mathrm{dom}(\mathsf{v}) = [1,4]$. The range encoding for v is

$$
\begin{array}{ll}
[\![\mathsf{v} \in [1,1]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [2,4]]\!] & [\![\mathsf{v} \in [1,2]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [3,4]]\!] \\
[\![\mathsf{v} \in [1,3]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [4,4]]\!] & [\![\mathsf{v} \in [1,4]]\!] \leftarrow \\
[\![\mathsf{v} \in [2,2]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [1,1]]\!],\ \mathit{not}\ [\![\mathsf{v} \in [3,4]]\!] & \\
[\![\mathsf{v} \in [2,3]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [1,1]]\!],\ \mathit{not}\ [\![\mathsf{v} \in [4,4]]\!] & \\
[\![\mathsf{v} \in [2,4]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [1,1]]\!] & \\
[\![\mathsf{v} \in [3,3]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [1,2]]\!],\ \mathit{not}\ [\![\mathsf{v} \in [4,4]]\!] & \\
[\![\mathsf{v} \in [3,4]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [1,2]]\!] & \\
[\![\mathsf{v} \in [4,4]]\!] \leftarrow \mathit{not}\ [\![\mathsf{v} \in [1,3]]\!] & \\
\leftarrow [\![\mathsf{v} \in [1,1]]\!], \mathit{not}\ [\![\mathsf{v} \in [1,2]]\!] & \leftarrow [\![\mathsf{v} \in [2,2]]\!], \mathit{not}\ [\![\mathsf{v} \in [1,2]]\!] \\
\leftarrow [\![\mathsf{v} \in [1,2]]\!], \mathit{not}\ [\![\mathsf{v} \in [1,3]]\!] & \leftarrow [\![\mathsf{v} \in [2,3]]\!], \mathit{not}\ [\![\mathsf{v} \in [1,3]]\!] \\
\leftarrow [\![\mathsf{v} \in [1,3]]\!], \mathit{not}\ [\![\mathsf{v} \in [1,4]]\!] & \leftarrow [\![\mathsf{v} \in [2,4]]\!], \mathit{not}\ [\![\mathsf{v} \in [1,4]]\!] \\
\leftarrow [\![\mathsf{v} \in [2,2]]\!], \mathit{not}\ [\![\mathsf{v} \in [2,3]]\!] & \leftarrow [\![\mathsf{v} \in [3,3]]\!], \mathit{not}\ [\![\mathsf{v} \in [2,3]]\!] \\
\leftarrow [\![\mathsf{v} \in [2,3]]\!], \mathit{not}\ [\![\mathsf{v} \in [2,4]]\!] & \leftarrow [\![\mathsf{v} \in [3,4]]\!], \mathit{not}\ [\![\mathsf{v} \in [2,4]]\!] \\
\leftarrow [\![\mathsf{v} \in [3,3]]\!], \mathit{not}\ [\![\mathsf{v} \in [3,4]]\!] & \leftarrow [\![\mathsf{v} \in [4,4]]\!], \mathit{not}\ [\![\mathsf{v} \in [3,4]]\!]
\end{array}
$$

From this, we can construct the set of nogoods $\Delta_{\text{ASP-RANGE}[\{\mathsf{v}\}]}$. The situation where no value has been removed from v's domain is represented in an assignment $\mathbf{A}$ such that $[\![\mathsf{v} \in [l,u]]\!] \notin \mathbf{A}^{\mathbf{T}} \cup \mathbf{A}^{\mathbf{F}}$ for $1 \leq l \leq u \leq 4$ except for $\mathbf{T}[\![\mathsf{v} \in [1,4]]\!] \in \mathbf{A}$. An updated range $[l,u]$ for v's domain is indicated by $\mathbf{T}[\![\mathsf{v} \in [l,u]]\!] \in \mathbf{A}$, for instance, $\mathbf{T}[\![\mathsf{v} \in [2,3]]\!] \in \mathbf{A}$ if the new range is $[2,3]$, and the channelling conditions also ensure $\mathbf{T}[\![\mathsf{v} \in [1,3]]\!], \mathbf{T}[\![\mathsf{v} \in [2,4]]\!] \in \mathbf{A}$. On the other hand, the exclusion of a range $[u,l]$ from v's domain is indicated by $\mathbf{F}[\![\mathsf{v} \in [l,u]]\!] \in \mathbf{A}$, for instance, $\mathbf{F}[\![\mathsf{v} \in [2,3]]\!] \in \mathbf{A}$ if the range $[2,3]$ is excluded. The channelling con-

ditions also ensure $\mathbf{F}[[\mathsf{v} \in [2,2]]], \mathbf{F}[[\mathsf{v} \in [3,3]]] \in \mathbf{A}$. The value of v is fixed to i when $\mathbf{T}[[\mathsf{v} \in [i,i]]] \in \mathbf{A}$. Some examples are given in the following table.

1	2	3	4	Property of $\mathbf{A}$
✓	✓	✓	✓	$\mathbf{T}[[\mathsf{v} \in [1,4]]] \in \mathbf{A}$
✓			✓	$\mathbf{T}[[\mathsf{v} \in [1,4]]], \mathbf{F}[[\mathsf{v} \in [2,3]]], \mathbf{F}[[\mathsf{v} \in [2,2]]], \mathbf{F}[[\mathsf{v} \in [3,3]]] \in \mathbf{A}$
		✓	✓	$\mathbf{T}[[\mathsf{v} \in [1,4]]], \mathbf{F}[[\mathsf{v} \in [1,2]]], \mathbf{T}[[\mathsf{v} \in [3,4]]], \mathbf{F}[[\mathsf{v} \in [1,1]]], \mathbf{F}[[\mathsf{v} \in [2,2]]] \in \mathbf{A}$
		✓		$\mathbf{T}[[\mathsf{v} \in [1,4]]], \mathbf{T}[[\mathsf{v} \in [1,3]]], \mathbf{T}[[\mathsf{v} \in [2,4]]], \mathbf{T}[[\mathsf{v} \in [2,3]]], \mathbf{T}[[\mathsf{v} \in [3,3]]] \in \mathbf{A}$

The table identifies values that have not been removed from the domain of v (✓) and some properties that can be observed in $\mathbf{A}$ (no complete list). Note that, different to the bound encoding, holes in the domain are reflected in $\mathbf{A}$.

Similar to the bound encoding, we encode constraints via conflict regions. We represent a conflict region $\mathsf{v}_1 \in [l_1, u_1], \ldots, \mathsf{v}_n \in [l_n, u_n]$ by a rule of the form

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 \in [l_1, u_1]]], \ldots, [[\mathsf{v}_n \in [l_n, u_n]]] \tag{4.14}$$

and repeat the process for each conflict region. The intuition behind a rule of the form (4.14) is similar to the one from the bound encoding. That is, whenever $l_1 \leq \mathsf{v}_1 \leq u_1, \ldots, l_n \leq \mathsf{v}_n \leq u_n$ holds, i.e., a conflict region is satisfied, then the constraint is violated.

Example 4.7

Reconsider the setting from Example 4.2, i.e., two variables v_1 and v_2 with domains $\mathrm{dom}(\mathsf{v}_1) = [2,4]$ and $\mathrm{dom}(\mathsf{v}_2) = [1,3]$, and the constraint $c = \mathsf{v}_1 \leq \mathsf{v}_2$. The range encoding of c is given through the following set of rules.

$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 \in [2,4]]],\ [[\mathsf{v}_2 \in [1,1]]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 \in [3,4]]],\ [[\mathsf{v}_2 \in [1,2]]]$$
$$[[\overline{c}]] \leftarrow [[\mathsf{v}_1 \in [4,4]]],\ [[\mathsf{v}_2 \in [1,3]]]$$

Verify that UP prunes the value 4 from v_1's domain and the value 1 from v_2's domain, and hence, achieves a range consistent assignment.

In general, UP on the range encoding achieves range consistency.

Theorem 4.5: Unit Propagation on the Range Encoding

UP on the range encoding enforces range consistency on the original constraint.

Proof. Let c be any constraint and $\mathbf{A}$ be an assignment to the atoms in a range representation of the variables in the scope of the constraint, i.e., the atoms in the program ASP-RANGE[scope(c)], representing the current ranges on the domain of the variables. Suppose UP has been run to completion, extending $\mathbf{A} \cup \{\mathbf{F}[[\overline{c}]]\}$ (i.e., the constraint c shall not be violated) to the conflict-free assignment $\mathbf{A}'$. We show by proof of contradiction that the domains are range consistent.

Suppose there is a variable $\mathsf{v}_i \in \text{scope}(c)$ such that if v_i is assigned a value between the minimum value l_i and the maximum value u_i of its domain, say $\mathsf{v}_i = k$ where $l_i \leq k \leq u_i$, then there is no bound support in the other variables $\mathsf{v}_1, \ldots, \mathsf{v}_{i-1}, \mathsf{v}_{i+1}, \ldots \mathsf{v}_n \in \text{scope}(c)$, i.e., there are no compatible values between their respective minimum $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n$ and their maximum values $u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n$. That means, all assignments are in a conflict region

$$l'_1 \leq l_1 \leq \mathsf{v}_1 \leq u_1 \leq u'_1, \ldots, l'_n \leq l_n \leq \mathsf{v}_n \leq u_n \leq u'_n$$

encoded by a rule r of the form (4.14).

For each v_j, where $1 \leq j \leq n$, we have $[[\mathsf{v}_j \in [l_j, u_j]]] \in (\mathbf{A}')^{\mathbf{T}}$ representing $\mathsf{v}_j \in [l_j, u_j]$. Since $\mathbf{A}'$ is conflict-free, nogoods represented by the channelling conditions (4.12) and (4.13), most importantly $\{\mathbf{T}[[\mathsf{v}_j \in [l_j, u_j]]], \mathbf{F}[[\mathsf{v}_j \in [l_j - 1, u_j]]]\}$ and $\{\mathbf{T}[[\mathsf{v}_j \in [l_j, u_j]]], \mathbf{F}[[\mathsf{v}_j \in [l_j, u_j + 1]]]\}$ guarantee that $[[\mathsf{v}_j \in [l'_j, u'_j]]] \in (\mathbf{A}')^{\mathbf{T}}$ for all intervals $[l_j, u_j] \subseteq [l'_j, u'_j]$. In turn, the nogood $\{\mathbf{F}\,\text{body}(r), \mathbf{T}[[\mathsf{v}_1 \in [l'_1, u'_1]]], \ldots, \mathbf{T}[[\mathsf{v}_n \in [l'_n, u'_n]]]\}$, represented by r (defined above), enforces $[[\mathsf{v}_i \in [l'_i, u'_i]]] \in (\mathbf{A}')^{\mathbf{F}}$ representing $\mathsf{v}_i \notin [l'_i, u'_i]$. Since $[k, k] \subseteq [l'_i, u'_i]$ and, again, by the nogoods represented in (4.12) and (4.13), we have $[[\mathsf{v}_i \in [k, k]]] \in (\mathbf{A}')^{\mathbf{F}}$. That means, the value k is not in the domain of v_i. This contradicts our assumption.

Hence, we conclude that the domains are range consistent as required. Since at least one value must be in each domain, guaranteed by (4.11), we have a set of non-empty domains which are range consistent. □

4.6 Mixed Encodings and Primitive Constraints

Encodings that mix various representations of the variables V's domains are also possible, for instance, using the atoms from the value and bound encoding, i.e.,

the atoms from ASP-VALUE[V] and ASP-BOUND[V]. Another example is mixing the representations from the bound and range encoding. While mixed encodings can increase space complexity, they have their advantages.

- An encoding of the variables' domains that makes use of more than just one representation does introduce a multitude of new atoms. On the other hand, such an encoding allows for the search heuristic of the ASP solvers decision algorithm to select atoms from either representation. With the success of search heuristics that are based on activity during the CONFLICT-ANALYSIS in a conflict-driven solver (cf. Goldberg and Novikov, 2002), this is of particular interest as the solver's search can dynamically adjust to a representation that produces results faster.

- It is inherent with some encodings, in particular the value encoding, that assignments made by the solver's search heuristic can represent a very strong commitment. For instance, it is often more beneficial to branch on bounds (*domain splitting*; Dincbas et al., 1988) and, in turn, prune values to approximate a solution, rather than to branch on exact values which may immediately lead to failure. On the other hand, some constraints may propagate better with a value encoding at hand.

- Ultimately, if a CASP encoding includes constraints with ASP decompositions that are best represented with different representations of variables, it is imperative to use mixed encodings.

Defining a mixed encoding is very easy. In the following, we will present two mixed encodings, the *value and bound encoding* and the *bound and range encoding*.

Value-by-bound Encoding

To achieve a mixed representation of the value and bounds of the variables' domains, we can take the bound encoding, i.e., ASP-BOUND[V], and combine it with rules of the form

$$[[\mathsf{v} = i]] \leftarrow [[\mathsf{v} \leq i]],\ not\ [[\mathsf{v} \leq i-1]] \tag{4.15}$$

where $v \in \mathsf{V}$ and $1 < i \leq d$. A rule of the form (4.15) defines an assignment to a value via bounds. The mixed value and bound representation introduces $\mathcal{O}(nd)$ new atoms and $\mathcal{O}(nd)$ rules, where $n = |\mathsf{V}|$ and d is the largest domain size.

Example 4.8

Reconsider the variable v with dom(v) = [1,4]. The mixed value and bound encoding for v is

$$\{[[\mathsf{v} \leq 1]], [[\mathsf{v} \leq 2]], [[\mathsf{v} \leq 3]], [[\mathsf{v} \leq 4]]\} \leftarrow$$

$$\leftarrow [[\mathsf{v} \leq 1]], not\ [[\mathsf{v} \leq 2]]$$

$$\leftarrow [[\mathsf{v} \leq 2]], not\ [[\mathsf{v} \leq 3]]$$

$$\leftarrow [[\mathsf{v} \leq 3]], not\ [[\mathsf{v} \leq 4]]$$

$$\leftarrow not\ [[\mathsf{v} \leq 4]]$$

$$[[\mathsf{v} = 1]] \leftarrow [[\mathsf{v} \leq 1]]$$

$$[[\mathsf{v} = 2]] \leftarrow not\ [[\mathsf{v} \leq 1]],\ [[\mathsf{v} \leq 2]]$$

$$[[\mathsf{v} = 3]] \leftarrow not\ [[\mathsf{v} \leq 2]],\ [[\mathsf{v} \leq 3]]$$

$$[[\mathsf{v} = 4]] \leftarrow not\ [[\mathsf{v} \leq 3]],\ [[\mathsf{v} \leq 4]]$$

From this, we can construct the set of nogoods $\Delta_{\text{ASP-VALUE-BY-BOUND}[\{\mathsf{v}\}]}$. The situation where no value has been removed from v's domain is represented in an assignment $\mathbf{A}$ such that $[[\mathsf{v} = i]], [[\mathsf{v} \leq i]] \not\in \mathbf{A^T} \cup \mathbf{A^F}$ for $i \in [1,4]$ except for $\mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$. The update of lower and upper bounds on the values of v works like in the bound encoding. The value of v is fixed to i when $\mathbf{F}[[\mathsf{v} \leq i-1]], \mathbf{T}[[\mathsf{v} \leq i]] \in \mathbf{A}$ and $\mathbf{T}[[\mathsf{v} = i]] \in \mathbf{A}$. This dependency is being taken care of by UP. Some examples are given in the table below.

1	2	3	4	Property of $\mathbf{A}$
✓	✓	✓	✓	$\mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}, [[\mathsf{v} = 1]], [[\mathsf{v} = 2]], [[\mathsf{v} = 3]], [[\mathsf{v} = 4]] \not\in \mathbf{A^T} \cup \mathbf{A^F}$
✓			✓	$\mathbf{F}[[\mathsf{v} = 2]], \mathbf{F}[[\mathsf{v} = 3]], \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}, [[\mathsf{v} = 1]], [[\mathsf{v} = 4]] \not\in \mathbf{A^T} \cup \mathbf{A^F}$
		✓	✓	$\mathbf{F}[[\mathsf{v} = 1]], \mathbf{F}[[\mathsf{v} = 2]], \mathbf{F}[[\mathsf{v} \leq 2]], \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$
		✓		$\mathbf{T}[[\mathsf{v} = 3]], \mathbf{F}[[\mathsf{v} \leq 2]], \mathbf{T}[[\mathsf{v} \leq 3]] \in \mathbf{A}$

The table identifies values that have not been removed from the domain of v (✓) and an some properties that can be observed in $\mathbf{A}$. Note that holes in the domain are reflected in $\mathbf{A}$.

Range-by-bound Encoding

Similarly, to achieve a mixed representation of the bounds and ranges of a variables' domains, we can take the bound encoding, i.e., ASP-BOUND[V], and com-

bine it with rules of the form

$$[[\mathsf{v} \in [l,u]]] \leftarrow [[\mathsf{v} \leq u]],\ not\ [[\mathsf{v} \leq l-1]] \tag{4.16}$$

where $v \in \mathsf{V}$ and $1 \leq l \leq u \leq d$. Here, a rule of the form (4.16) maps bounds on the domain of a variable to a range. We denote this encoding ASP-RANGE-BY-BOUND[V]. In total, the mixed bound and range representation introduces $\mathcal{O}(nd^2)$ new atoms and $\mathcal{O}(nd^2)$ rules, where $n = |\mathsf{V}|$ and d is the largest domain size.

Example 4.9

Reconsider the variable v with dom(v) = [1,4]. The mixed bound and range encoding for v is

$$\{[[\mathsf{v} \leq 1]], [[\mathsf{v} \leq 2]], [[\mathsf{v} \leq 3]], [[\mathsf{v} \leq 4]]\} \leftarrow$$

$$\leftarrow [[\mathsf{v} \leq 1]], not\ [[\mathsf{v} \leq 2]]$$

$$\leftarrow [[\mathsf{v} \leq 2]], not\ [[\mathsf{v} \leq 3]]$$

$$\leftarrow [[\mathsf{v} \leq 3]], not\ [[\mathsf{v} \leq 4]]$$

$$\leftarrow not\ [[\mathsf{v} \leq 4]]$$

$$[[\mathsf{c} \in [1,1]]] \leftarrow [[\mathsf{v} \leq 1]] \qquad [[\mathsf{c} \in [1,2]]] \leftarrow [[\mathsf{v} \leq 2]]$$

$$[[\mathsf{c} \in [1,3]]] \leftarrow [[\mathsf{v} \leq 3]] \qquad [[\mathsf{c} \in [1,4]]] \leftarrow [[\mathsf{v} \leq 4]]$$

$$[[\mathsf{c} \in [2,2]]] \leftarrow not\ [[\mathsf{c} \leq 1]],\ [[\mathsf{v} \leq 2]]$$

$$[[\mathsf{c} \in [2,3]]] \leftarrow not\ [[\mathsf{c} \leq 1]],\ [[\mathsf{v} \leq 3]]$$

$$[[\mathsf{c} \in [2,4]]] \leftarrow not\ [[\mathsf{c} \leq 1]],\ [[\mathsf{v} \leq 4]]$$

$$[[\mathsf{c} \in [3,3]]] \leftarrow not\ [[\mathsf{c} \leq 2]],\ [[\mathsf{v} \leq 3]]$$

$$[[\mathsf{c} \in [3,4]]] \leftarrow not\ [[\mathsf{c} \leq 2]],\ [[\mathsf{v} \leq 4]]$$

$$[[\mathsf{c} \in [4,4]]] \leftarrow not\ [[\mathsf{c} \leq 3]],\ [[\mathsf{v} \leq 4]]$$

From this, we can construct the set of nogoods $\Delta_{\text{ASP-RANGE-BY-BOUND}[\{\mathsf{v}\}]}$. The situation where no value has been removed from v's domain is represented in an assignment $\mathbf{A}$ such that $[[\mathsf{v} \leq i]] \notin \mathbf{A^T} \cup \mathbf{A^F}$ for $i \in [1,3]$ and $[[\mathsf{v} \in [l,u]]] \notin \mathbf{A^T} \cup \mathbf{A^F}$ for $1 \leq l \leq u \leq 4$ except for $\mathbf{T}[[\mathsf{v} \in [1,4]]] \in \mathbf{A}$. The update of lower and upper bounds on the values of v works like in the bound encoding, but also reflected by an updated range $[l,u]$ for v's domain. This dependency is being taken care of by UP. For instance, $\mathbf{F}[[\mathsf{v} \leq 1]], \mathbf{T}[[\mathsf{v} \leq 3]], \mathbf{T}[[\mathsf{v} \in [2,3]]] \in \mathbf{A}$ if the new lower

bound is 2 and the new upper bound is 3. Some more examples are given in the table below.

1	2	3	4	Property of **A**
✓	✓	✓	✓	$\mathbf{T}[[\mathsf{v} \in [1,4]]], \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$
✓			✓	$\mathbf{T}[[\mathsf{v} \in [1,4]]], \mathbf{T}[[\mathsf{v} \leq 4]], \mathbf{F}[[\mathsf{v} \in [2,3]]] \in \mathbf{A}$
		✓	✓	$\mathbf{F}[[\mathsf{v} \in [1,2]]], \mathbf{F}[[\mathsf{v} \leq 2]], \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}$
		✓		$\mathbf{F}[[\mathsf{v} \in [1,2]]], \mathbf{F}[[\mathsf{v} \leq 2]], \mathbf{T}[[\mathsf{v} \leq 3]], \mathbf{T}[[\mathsf{v} \leq 4]] \in \mathbf{A}, \mathbf{T}[[\mathsf{v} \in [3,3]]] \in \mathbf{A}$

The table identifies values that have not been removed from the domain of v (✓) and some properties that can be observed in **A** (no complete list). Note that holes in the domain are not reflected in **A**.

Range Support Encoding

In principle, we can define a support encoding based on the bound or range representation of the variables' domains.

A *(binary) range support* for a variable v to take a value from the range $[l, u]$ across a constraint $c \in \mathsf{C}$ is the set of domain ranges $[l_1, u_1], \ldots, [l_m, u_m]$ of another variable in $\mathsf{v}' \in \text{scope}(c) \setminus \{\mathsf{v}\}$, where $u_i < l_{i+1}$ for $1 \leq i < m$, that do not violate the constraint if $\mathsf{v} \in [l_i, u_i]$. Let $c \in \mathsf{C}$ and $\text{scope}(c) = (\mathsf{v}_1, \ldots, \mathsf{v}_n)$. To achieve a support encoding of c that works on the range representation of the domains of the variables in its scope, we encode range support by a rule of the form

$$[[\overline{c}]] \leftarrow [[\mathsf{v} \in [l, u]]], not\ [[\mathsf{v}' \in [l_1, u_1]]], \ldots, not\ [[\mathsf{v}' \in [l_m, u_m]]] \tag{4.17}$$

where the set of domain ranges $[l_1, u_1], \ldots, [l_m, u_m]$ for variable v' form the range support for $\mathsf{v} \in [l, u]$ across c. A rule of the form (4.17) can be read as whenever $\mathsf{v} \in [l, u]$, then at least one of its range supports must hold, otherwise the constraint is violated. In the range support encoding of a constraint $c \in \mathsf{C}$, we encode range support for each pair of distinct variables $\mathsf{v}, \mathsf{v}' \in \text{scope}(c)$, and for each range $[l, u] \subseteq \text{dom}(\mathsf{v})$.

Primitive Constraints

A last remark concerns a simple optimisation for translating CASP to ASP. While we have introduced atoms of the form $[[\mathsf{v} = i]]$, $[[\mathsf{v} \leq i]]$ and $[[\mathsf{v} \in [l, u]]]$ as value, bound, and range representations of the variable v's domain, respectively, they can also be seen as *primitive constraints*. Different from decomposing constraints

via forbidden combination of values or regions, a primitive constraint does not require decomposition. Instead, the corresponding variable encoding can be used directly, without introducing auxiliary constraint atoms.

Example 4.10

Consider any constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ such that $\mathsf{v} \dot{\leq} i \in \mathsf{C}$ is the constraint that the value of $\mathsf{v} \in \mathsf{V}$ must be smaller or equal i, and Π contains the rule r

$$p \leftarrow q,\ [\![\mathsf{v} \dot{\leq} i]\!]$$

where $[\![\mathsf{v} \dot{\leq} i]\!]$ is a constraint atom representing $\mathsf{v} \dot{\leq} i$, i.e., $\mathrm{constraint}([\![\mathsf{v} \dot{\leq} i]\!]) = \mathsf{v} \dot{\leq} i$. (We use $\mathsf{v} \dot{\leq} i$ to distinguish the constraint atom $[\![\mathsf{v} \dot{\leq} i]\!]$ from the bounds representation of v's domain.) The bound encoding of $\mathsf{v} \dot{\leq} i$ is given by the rules

$$[\![\overline{\mathsf{v} \dot{\leq} i}]\!] \leftarrow not\ [\![\mathsf{v} \leq i]\!]$$
$$[\![\mathsf{v} \dot{\leq} i]\!] \leftarrow not\ [\![\overline{\mathsf{v} \dot{\leq} i}]\!]$$

where $[\![\mathsf{v} \leq i]\!]$ is from the bound representation of v's domain.

We can substitute above encoding by simply replacing every occurrence of $[\![\mathsf{v} \dot{\leq} i]\!]$ in Π with $[\![\mathsf{v} \leq i]\!]$ in Π. The rule f for instance, changes to

$$p \leftarrow q,\ [\![\mathsf{v} \leq i]\!].$$

Similarly, in the bound encoding of the constraint $\mathsf{v} \dot{>} i \in \mathsf{C}$ that the value of $\mathsf{v} \in \mathsf{V}$ must be strictly greater than i, the constraint atom $[\![\mathsf{v} \dot{>} i]\!]$ may be substituted with the default negated atom $not\ [\![\mathsf{v} \leq i]\!]$.

Whilst this optimisation is worthwhile in a practical implementation of our approach, note that it goes slightly beyond Definition 4.1.

4.7 Related Work

ASP was put forward as a paradigm for modelling and solving CSP by Niemelä (1999). Niemelä also presented a systematic, straightforward mapping from CSP to ASP, including an encoding of constraints via either allowed or forbidden combination of values. This is reflected in our value encoding. We have demonstrated in Section 4.2 that this approach can hinder propagation, e.g., arc consistency on the original constraint cannot be achieved through UP. A consequence of the work

by You and Hou (2004) is that this gap can be closed by employing lookahead, i.e., extending propagation with failed-literal detection. Whilst UP runs in linear-time, using lookahead has quadratic costs in the size of the encoding. We devised support, bound, and range encodings as alternatives in Sections 4.3–4.5 that only require UP.

Decompositions of constraints have been previously studied in the related area of SAT. Walsh (2000) analysed two different mappings of binary constraints into SAT, i.e., the *direct encoding* and the *log encoding*. While the direct encoding is reflected by our value encoding, the log encoding represents the bit-vector of each variable. Although the log encoding is more space-efficient, UP on the direct encoding prunes more possible values than UP on the log encoding (Walsh, 2000). Gent (2002) proposed to encode *support* rather than encoding conflicts, and showed that UP can maintain arc consistency on the original constraint. This technique was generalised to relational k-arc consistency by Bessière et al. (2003).

When encoding CSP into SAT, the condition that a variable cannot take more than one value is sometimes ignored when it is not necessary for proving the existence of a solution. This may be of benefit in terms of space complexity, as naïve SAT encodings of the at least-one condition requires quadratic space in the size of the domain. Though, linear-size encodings that introduce auxiliary atoms exist (cf. Sinz, 2005). Similarly, in our context, we can drop the cardinality constraint rule (4.3) from the value encoding. In effect, the one-to-one correspondence between the answer sets of a CSP's ASP encodings and the CSP's configurations is weakened (cf. Definition 4.1): If an answer set is found that represents an assignment of two values to a variable v then a configuration exists.

Apart from the direct encoding, representing bounds on the variables' domains is also common in the area of SAT. Using this method, CSPs have been successfully solved with the award-winning system *sugar* (Tamura et al., 2006).

To our knowledge, the only related work that represents domain ranges in a Boolean variable was conducted by Bessière et al. (2009a) in their decomposition of the ALL-DIFFERENT constraint into cardinality constraints. We will follow their idea when we present a specialised ASP encoding for ALL-DIFFERENT in the next chapter.

4.8 Conclusions

In this chapter, we have presented a translation-based approach to constraint answer set solving. We began with laying the theoretical foundations of the key idea, i.e., to compile the variables' domains and constraints of a constraint program into ASP. By exploiting Lifschitz and Turner's splitting set theorem, the problem of computing the constraints answer sets of a constraint program can then be reduced to computing the answer sets of a single, joint ASP encoding for which efficient off-the-shelf solvers are available, in particular, conflict-driven solvers.

We then started with translating CSP into ASP by presenting various generic encodings of constraints based on the value, bound, range, or mixed representation of the variables' domains, and investigated the level of local consistency ASP inference can maintain on the original constraint. I.e., applying UP on the generic support, bound, or range encoding achieves arc, bound, or range consistency, respectively.

The applicability of our technique is not limited to CASP: To our knowledge, previous work on modelling with ASP was not concerned with the propagation strength of their encodings. The idea of studying the effect of ASP inference in terms of local consistency on the condition encoded in ASP constructs might add to understanding the interaction between formulation and algorithm in ASP in general.

We will proceed with our translation-based approach to constraint answer set solving by investigating more specialised ASP encodings for important global constraints in the next chapter.

Chapter 5

Encoding Global Constraints with Answer Set Programming

In the previous chapter, we have presented a translation-based approach to constraint answer set solving. In particular, we have studied four generic encodings: the value, support, bound, and range encoding. In turn, we demonstrated that the UP inference of any ASP solver on these encodings provides a propagator of the original constraint that can hinder propagation, or achieve arc, bound, and range consistency, respectively.

We here proceed with our investigation by considering more specialised, yet simple encodings for important global constraints, that are, ALL-DIFFERENT (van Hoeve, 2001), GRAMMAR and related constraints (Pesant, 2004; Sellmann, 2006; Quimper and Walsh, 2006), and REACHABILITY (Dooms et al., 2005). We shall see that the ASP inference on our encodings can simulate the effect of complex propagation algorithms from CP with similar asymptotic run time complexity.

5.1 The ALL-DIFFERENT Constraint

We start with ALL-DIFFERENT (van Hoeve, 2001), one of the most important (global) constraints. Because of its importance, ALL-DIFFERENT represents a crucial benchmark for our translation-based approach to constraint answer set solving. We make several contributions to this line of research.

– First, we demonstrate that the ALL-DIFFERENT constraint can be modelled straightforwardly with ASP using cardinality constraint rules. We show that,

for this encoding, UP inference maintains arc consistency, and provide theoretical results on asymptotic run time complexity.

– Then, we present alternative ASP encodings based on a range or bound representation of the variables' domains. Again, we prove their properties in terms of local consistency achieved by UP and its run time complexity. In particular, the UP inference of any ASP solver can simulate the effect of complex constraint propagation algorithms like the one in (Leconte, 1996).

– Finally, we demonstrate applicability on problems from the CSPLib (Gent and Walsh, 1999), a large problem library widely used for benchmarking in CP. Our results demonstrate the competitiveness of our approach.

The effect of ALL-DIFFERENT is that a set of variables take all different values.

Definition 5.1: ALL-DIFFERENT

Given an assignment A, the constraint ALL-DIFFERENT$(\{\mathsf{v}_1, \ldots, \mathsf{v}_n\})$ is satisfied if and only if $A(\mathsf{v}_i) \neq A(\mathsf{v}_j)$ for $1 \leq i < j \leq n$.

A key concept in propagating ALL-DIFFERENT is the notion of a Hall set (Hall, 1935). This is a set of m values which completely contains the domains of $\geq m$ variables. Given an ALL-DIFFERENT constraint c, H is a *Hall set* if $|\{\mathsf{v} \in \text{scope}(c) \mid \text{dom}(\mathsf{v}) \subseteq H\}| \geq |H|$. If the strict inequality holds, then c is violated. If the equality holds, then in any domain consistent assignment, the variables whose domains are contained in the Hall set consume all the values in the Hall set, whilst other variables must find values outside the Hall set.

It is not practical to check every single one of the $2^{|d|}$ sets for their Hall property. A propagator for ALL-DIFFERENT that determines Hall sets in polynomial time exists (Régin, 1994), but it cannot be simulated efficiently with UP (Bessière et al., 2009b).

Arc Consistent Encoding of ALL-DIFFERENT

In order to achieve an arc consistent assignment in polynomial time, we can limit ourselves to Hall sets of size 1, that are, represented by the domain of a variable whose value is fixed. All other variables must find values outside the Hall set, in any arc consistent assignment.

Example 5.1

Consider an ALL-DIFFERENT constraint over the variables $\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4$ with the

following domains:

	1	2	3	4
v_1		✓		
v_2	✓	✓	✓	
v_3	✓	✓	✓	✓
v_4		✓	✓	

$\{2\}$ is a Hall set of size 1 as the domain of a single variable, v_1, is completely contained within it. Therefore, the value 2 can be removed from the domains of all the other variables. This leaves the variable v_4 with the value 3, and we obtain another Hall set of size 1, i.e., the set $\{3\}$. Removing 3 from the domains of the other variables leaves v_2 with the value 1. In turn, this new Hall set leads to 4 being the only remaining value of v_4.

As no further pruning is possible, this leaves the following arc consistent domains:

	1	2	3	4
v_1		✓		
v_2	✓			
v_3				✓
v_4			✓	

We can use cardinality constraint rules to express the condition induced by Hall sets of size 1 straightforwardly into ASP.

Definition 5.2: ASP Encoding of ALL-DIFFERENT Constraints

Our ASP encoding of the ALL-DIFFERENT$(\{v_1,\ldots,v_n\})$ constraint is

$$[[\overline{\text{ALL-DIFFERENT}}(\{v_1,\ldots,v_n\})]] \leftarrow 2\ \{\ [[v_1 = i]]\ldots,[[v_n = i]]\ \} \tag{5.1}$$

where $1 \leq i \leq d$. We denote the encoding by ASP-ALL-DIFFERENT$[\{v_1,\ldots,v_n\}]$.

Observe that our encoding of ALL-DIFFERENT introduces only $\mathcal{O}(d)$ rules, is quite simple, yet ASP inference enforces arc consistency.

Theorem 5.1: Arc Consistency on ALL-DIFFERENT via Unit Propagation

UP on ASP-ALL-DIFFERENT$[\{v_1,\ldots,v_n\}]$ $\cup$ ASP-VALUE$[\{v_1,\ldots,v_n\}]$ enforces arc consistency on the binary decomposition of ALL-DIFFERENT$(\{v_1,\ldots,v_n\})$.

Proof. The proof strategy is to show that if an assignment to a variable is possible then, for each other variable, there is a value in the domain that satisfies the binary decomposition of the constraint.

Let $\mathbf{A}$ be an assignment to the atoms in a value representation of the variables in the scope of the constraint, i.e., the atoms in the program ASP-VALUE$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$, representing the current set of values in the domain of the variables. Suppose UP has been run to completion, extending $\mathbf{A} \cup \{\mathbf{F}[[\overline{\text{ALL-DIFFERENT}}(\{\mathsf{v}_1,\ldots,\mathsf{v}_n\})]]$ (i.e., the ALL-DIFFERENT constraint shall not be violated) to the assignment $\mathbf{A}'$. We show by proof of contradiction that the domains are arc consistent.

Suppose there is a variable v_i in the scope of the constraint and a value k that has not been removed from its domain, and there is some other variable v_j where $1 \leq j \leq n$ with $j \neq i$, that has no compatible value, i.e., the domains are not arc consistent. In particular, we have $[[\mathsf{v}_i = k]] \notin (\mathbf{A}')^{\mathbf{F}}$ whilst $[[\mathsf{v}_j = l]] \in (\mathbf{A}')^{\mathbf{F}}$ for $1 \leq l \leq d$ where $l \neq k$. Then, the nogood $\{\mathbf{F}[[\mathsf{v}_j = l]] \mid 1 \leq l \leq d\}$ represented by (4.2) guarantees $[[\mathsf{v}_j = k]] \in (\mathbf{A}')^{\mathbf{T}}$, and we have that the corresponding cardinality constraint rules (5.1) in ASP-ALL-DIFFERENT$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ ensure $[[\mathsf{v}_i = k]] \in (\mathbf{A}')^{\mathbf{F}}$. That means, the value k is not in the domain of v_i. This contradicts our assumption.

Hence, we conclude that the domains are arc consistent as required. Since at least one value must be in each domain, guaranteed by (4.2), we have a set of non-empty domains which are arc consistent. □

We now address run time complexity.

Theorem 5.2

UP on ASP-ALL-DIFFERENT$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}] \cup$ ASP-VALUE$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ runs in $\mathcal{O}(nd)$ time down any branch of the search tree.

Proof. For each of the n variables, there are $\mathcal{O}(d)$ constant-size nogoods and one nogood of size $\mathcal{O}(d)$ represented by ASP-VALUE$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$. On the other hand, for each of the d values, there is one cardinality constraint rule containing n atoms. By Theorems 2.1 and 2.6 we obtain, down any branch of the search tree, a total running time complexity given by $\mathcal{O}(nd) + \mathcal{O}(nd) = \mathcal{O}(nd)$. □

Notice that the size of an individual ASP construct like cardinality constraint rules can be more or equally relevant for the complexity of propagation than the number of rules in the entire encoding.

Range Consistent Encoding of ALL-DIFFERENT

A propagator for the ALL-DIFFERENT constraint that enforces range consistency pruning Hall intervals has been proposed by Leconte (1996). An interval $[l, u]$ is a *Hall interval* if $[l, u]$ is a Hall set, i.e., for an ALL-DIFFERENT constraint c we have $|\{\mathsf{v} \in \text{scope}(c) \mid \text{dom}(\mathsf{v}) \subseteq [l, u]\}| \geq |u - l + 1|$. In other words, a Hall interval $[l, u]$ completely contains the domains of $\geq u - l + 1$ variables. If the equality holds, then in any range consistent assignment, the variables whose domains are contained in the Hall interval consume all the values in the Hall interval, whilst other variables must find values outside the Hall interval.

We illustrate the effect of Hall intervals in the following example.

Example 5.2

Consider an ALL-DIFFERENT constraint over the variables $\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4$ with the following bound consistent domains:

	1	2	3	4
v_1		✓	✓	
v_2	✓	✓	✓	✓
v_3	✓	✓	✓	✓
v_4		✓	✓	

$[2, 3]$ is a Hall interval of size 2 as the domains of two variables, v_1 and v_4, are completely contained within it. We can thus remove $[2, 3]$ from the domains of v_2 and v_3. This leaves the following range consistent domains:

	1	2	3	4
v_1		✓	✓	
v_2	✓			✓
v_3	✓			✓
v_4		✓	✓	

As an alternative to a specialised propagator for ALL-DIFFERENT like Leconte's, Bessière et al. (2009a) proposed the decomposition of ALL-DIFFERENT into cardinality constraints that check whether no interval $[l, u]$ contains more variables than its size. Exploiting Hall intervals, Bessière et al. showed that maintaining domain consistency on their decomposition enforces range consistency on the original constraint.

Fortunately, most ASP systems natively support cardinality constraint rules. Following the idea of Bessière et al., we propose a simple ASP encoding based on our range representation of the variables' domains from the previous chapter.

Definition 5.3: Range Consistent ASP Encoding of ALL-DIFFERENT Constraints

Our range consistent ASP encoding of the ALL-DIFFERENT($\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$) constraint c is

$$[[\overline{c}]] \leftarrow u - l + 2 \; \{ \; [[\mathsf{v}_1 \in [l,u]]], \; \ldots, \; [[\mathsf{v}_n \in [l,u]]] \; \} \tag{5.2}$$

where $1 \leq l \leq u \leq d$. We denote it by ASP-ALL-DIFFERENT-RC[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$].

This simple encoding of ALL-DIFFERENT permits us to simulate the effect of complex constraint propagation algorithms like the one in (Leconte, 1996).

Theorem 5.3: Range Consistency on ALL-DIFFERENT via Unit Propagation

UP on ASP-ALL-DIFFERENT-RC[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$] $\cup$ ASP-RANGE[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$] enforces range consistency on ALL-DIFFERENT($\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$).

Proof. Let $\mathbf{A}$ be an assignment to the atoms in a range representation of the variables in the scope of the constraint, i.e., the atoms ASP-RANGE[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$], representing the current ranges on the domain of the variables. Suppose UP has been run to completion, extending $\mathbf{A} \cup \{\mathbf{F}[[\overline{\text{ALL-DIFFERENT}}(\{\mathsf{v}_1,\ldots,\mathsf{v}_n\})]]\}$ (i.e., the ALL-DIFFERENT constraint shall not be violated) to the conflict-free assignment $\mathbf{A}'$. Leconte (1996) has shown that the ALL-DIFFERENT constraint is range consistent if and only if every variable whose domains are not fully contained within a Hall interval has no value from the Hall interval it its domain. We show by proof of contradiction that the domains are range consistent.

Suppose there is a variable v_i in the scope of the constraint, such that if v_i is assigned a value between the minimum value l_i and the maximum value u_i of its domain, say $\mathsf{v}_i = k$ where $l_i \leq k \leq u_i$, then there is no bound support in the other variables $\mathsf{v}_1,\ldots,\mathsf{v}_{i-1},\mathsf{v}_{i+1},\ldots\mathsf{v}_n$, i.e., there are no compatible values between their respective minimum $l_1,\ldots,l_{i-1},l_{i+1},\ldots,l_n$ and their maximum values $u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n$. In particular, there is a Hall interval $[l,u]$ such that $1 \leq l \leq k \leq u \leq d$ and for the variables with indices $H = \{j \mid [[\mathsf{v}_j \in [l,u]]] \in (\mathbf{A}')^{\mathbf{T}},\ i \neq j\}$ we have $|H| = u-l+1$, i.e., v_i is outside of the Hall interval. The truth of $[[\mathsf{v}_j \in [l,u]]]$ for each $i \in H$ in guaranteed by nogoods represented by (4.12) and (4.13) because $\mathbf{A}'$ is conflict-free, and hence, the truth of smaller domain ranges, say $[l_j,u_j] \subseteq [l,u]$ was

channelled accordingly. Since $[[\overline{\text{ALL-DIFFERENT}}(\{\mathsf{v}_1,\ldots,\mathsf{v}_n\})]] \in (\mathbf{A}')^{\mathbf{F}}$, however, we have that the corresponding cardinality constraint rule (5.2) ensures $[[\mathsf{v}_\ell \in [l,u]]] \in (\mathbf{A}')^{\mathbf{F}}$ for each $\ell \notin H$ where $1 \leq \ell \leq n$. In particular, $[[\mathsf{v}_i \in [l,u]]] \in (\mathbf{A}')^{\mathbf{F}}$. Again, by the nogoods represented through (4.12) and (4.13), we have $[[\mathsf{v}_i \in [k,k]]] \in (\mathbf{A}')^{\mathbf{F}}$. That means, the value k is not in the domain of v_i. This contradicts our assumption.

Hence, we conclude that the domains are range consistent as required. Since at least one value must be in each domain, guaranteed by (4.11), we have a set of non-empty domains which are range consistent. □

In fact, ASP inference on our encoding has the same asymptotic run time complexity as Leconte's specialised algorithm.

Theorem 5.4

UP on ASP-ALL-DIFFERENT-RC$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ ∪ ASP-RANGE$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ runs in $\mathcal{O}(nd^2)$ time down any branch of the search tree.

Proof. For each of the n variables, there are $\mathcal{O}(d^2)$ nogoods of constant size represented by ASP-RANGE$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$. On the other hand, for each of the d^2 Hall intervals, there is one cardinality constraint rule in ASP-ALL-DIFFERENT-RC$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ containing n atoms. By Theorems 2.1 and 2.6 we obtain, down any branch of the search tree, a total running time complexity given by $\mathcal{O}(nd^2) + \mathcal{O}(nd^2) = \mathcal{O}(nd^2)$. □

Bound Consistent Encoding of ALL-DIFFERENT

In order to achieve a decomposition of ALL-DIFFERENT that can only prune bounds, Bessière et al. (2009a) proposed constraints that are woken whenever the bounds on the variables' domains change, and showed that maintaining bounds consistency on the resulting reformulation enforces bounds consistency on the original constraint. To reflect their idea within our context, we make a simple change to the way domains are represented. Recall that the bound encoding of the variables' domains can be mixed with a range representation, as given through RANGE-BY-BOUND that we have introduced in the previous chapter.

Then, UP on ASP-ALL-DIFFERENT-RC$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ has a different effect.

Theorem 5.5: Bound Consistency on ALL-DIFFERENT via Unit Propagation

UP on ASP-ALL-DIFFERENT-RC$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ ∪ ASP-RANGE-BY-BOUND$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$ enforces bound consistency on ALL-DIFFERENT$(\{\mathsf{v}_1,\ldots,\mathsf{v}_n\})$.

Proof. The proof follows the ones of Theorems 4.4 and 5.3, as the decompositions for range and bound consistency both encode the same conflict regions.

Let $\mathbf{A}$ be an assignment to the atoms in a mixed bound and range representation of the variables in the scope of the constraint, i.e., the atoms in the program ASP-RANGE-BY-BOUND$[\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}]$, representing the current domains of the variables. Suppose UP has been run to completion, extending the assignment $\mathbf{A} \cup \{\mathbf{F}[[\overline{\text{ALL-DIFFERENT}}(\{\mathsf{v}_1,\ldots,\mathsf{v}_n\})]]\}$ (i.e., the ALL-DIFFERENT constraint shall not be violated) to the conflict-free assignment $\mathbf{A}'$. We show by proof of contradiction that the domains are bound consistent.

Suppose there is a variable v_i, where $1 \leq i \leq n$, such that if v_i is assigned its minimum value l_i or its maximum value u_i, then there are no compatible values of the other variables $\mathsf{v}_1,\ldots,\mathsf{v}_{i-1},\mathsf{v}_{i+1},\ldots,\mathsf{v}_n$ between their respective minimum $l_1,\ldots,l_{i-1},l_{i+1},\ldots,l_n$ and their maximum values $u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n$, i.e., the domains are not bound consistent.

First, we analyse the case $\mathsf{v}_i = u_i$, i.e., v_i is assigned its maximum value u_i. Then, there is a Hall interval $[l,u]$ such that $1 \leq l \leq u_i \leq u \leq d$ and for the variables with indices $H = \{j \mid [[\mathsf{v}_j \in [l,u]]] \in (\mathbf{A}')^{\mathbf{T}},\ i \neq j\}$ we have $|H| = u - l + 1$, i.e., v_i is outside of the Hall interval. The assignments $\mathbf{T}[[\mathsf{v}_j \leq u]]$ and $\mathbf{F}[[\mathsf{v}_j \leq l-1]]$ for each $j \in H$ are guaranteed by nogoods represented by (4.8) because $\mathbf{A}'$ is conflict-free, and hence, the truth value of tighter bounds on the domains, say $l \leq l_j \leq u_j \leq u$, was channelled accordingly. In turn, nogoods $\{\mathbf{F}\,\text{body}(r), \mathbf{T}[[\mathsf{v}_j \leq u]], \mathbf{F}[[\mathsf{v}_j \leq l-1]]\}$ and $\{\mathbf{F}[[\mathsf{v}_j \in [l,u]]], \mathbf{T}\,\text{body}(r)\}$ represented by rules r of the form (4.16) guarantee $[[\mathsf{v}_j \in [l,u]]] \in (\mathbf{A}')^{\mathbf{T}}$. Since $[[\overline{\text{ALL-DIFFERENT}}(\{\mathsf{v}_1,\ldots,\mathsf{v}_n\})]] \in (\mathbf{A}')^{\mathbf{F}}$, however, we have that the corresponding cardinality constraint rule (5.2) ensures $[[\mathsf{v}_\ell \in [l,u]]] \in (\mathbf{A}')^{\mathbf{F}}$ for each $\ell \notin H$ where $1 \leq \ell \leq n$. In particular, $[[\mathsf{v}_i \in [l,u]]] \in (\mathbf{A}')^{\mathbf{F}}$. Observe that we have $[[\mathsf{v}_i \leq l-1]] \in (\mathbf{A}')^{\mathbf{F}}$ because $[[\mathsf{v}_i \leq l_i - 1]] \in (\mathbf{A}')^{\mathbf{F}}$. Then, the nogoods represented by (4.16) guarantee $[[\mathsf{v}_i \leq u]] \in (\mathbf{A}')^{\mathbf{F}}$, and through channelling (4.8) we get $[[\mathsf{v}_i \leq u_i]] \in (\mathbf{A}')^{\mathbf{F}}$, i.e., the value u_i is not in the domain of v_i. This contradicts our assumption.

The second case, where v_i is assigned its minimum value l_i, is symmetric.

Hence, we conclude that the domains are bound consistent as required. Since at least one value must be in each domain, guaranteed by (4.9), we have a set of non-empty domains which are range consistent. □

We now address run time complexity.

Theorem 5.6

UP on ASP-ALL-DIFFERENT-RC[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$]$\cup$ASP-RANGE-BY-BOUND[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$] runs in $\mathcal{O}(nd^2)$ time down any branch of the search tree.

Proof. For each of the n variables, there are $\mathcal{O}(d^2)$ nogoods of constant size represented by ASP-RANGE-BY-BOUND[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$]. On the other hand, for each of the d^2 Hall intervals, there is one cardinality constraint rule containing n atoms. By Theorems 2.1 and 2.6 we obtain, down any branch of the search tree, a total running time complexity given by $\mathcal{O}(nd^2)+\mathcal{O}(nd^2)=\mathcal{O}(nd^2)$. □

Although our range and bound consistency encodings of ALL-DIFFERENT have the same worst-case space complexity, i.e., they share the rules of the form (5.2), Theorems 5.3 and 5.5 report different effects from UP. This is due the way ranges are encoded. For instance, it is easy to see that UP on ASP-RANGE-BY-BOUND[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$] does not propagate gaps in the domains, whilst ASP-RANGE[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$] does. On the other hand, the former contains only $\mathcal{O}(nd)$ rules, compared to the $\mathcal{O}(nd^2)$ rules of the latter.

We can further limit the size of our decompositions by using an upper bound h on the size of Hall intervals. The resulting encoding with only those cardinality constraint rules (5.2) for which $u-l+1\leq h$ detects Hall intervals of size at most h. This will result in a smaller encoding, in exchange of achieving a weaker level of consistency.

Experimental Results

To test the competitiveness of our translation-based approach to CASP solving, we have conducted experiments on hard combinatorial problems that are traditionally modelled with ALL-DIFFERENT and PERMUTATION constraints. The PERMUTATION constraint is a special case of ALL-DIFFERENT when the number of variables is equal to the number of all their possible values. An ASP decomposition of the constraint $c=$ PERMUTATION($\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$) extends ASP-ALL-DIFFERENT[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$] by

$$[[\overline{c}]] \leftarrow not\ [[\mathsf{v}_1=i]],\ \ldots,\ not\ [[\mathsf{v}_n=i]] \tag{5.3}$$

and ASP-ALL-DIFFERENT-RC[$\{\mathsf{v}_1,\ldots,\mathsf{v}_n\}$] by rules of the form

$$[[\overline{c}]] \leftarrow d-u+l\ \{\ not\ [[\mathsf{v}_1\in[l,u]]],\ \ldots,\ not\ [[\mathsf{v}_n\in[l,u]]]\ \} \tag{5.4}$$

where $1\leq l\leq u\leq k$. This can increase propagation.

Example 5.3

Consider a PERMUTATION constraint over the variables v_1, v_2, v_3, v_4 with the following domains:

	1	2	3	4
v_1	✓			✓
v_2	✓	✓	✓	✓
v_3	✓	✓	✓	✓
v_4	✓			✓

Observe that there is no Hall interval (of size < 4). However, the effect of PERMUTATION beyond ALL-DIFFERENT, respectively the addition of cardinality constraint rules of the form (5.3) or (5.4), is that the Hall property is also maintained for intervals of removed values. For instance, $[2,3]$ is such an interval of size 2 as the domains of two variables, v_1 and v_3, does not include any value from it. Hence, the domains of the remaining variables must be contained in $[2,3]$. This leaves the following domains:

	1	2	3	4
v_1	✓			✓
v_2		✓	✓	
v_3		✓	✓	
v_4	✓			✓

Benchmark problems for our experiments stem from the CSPLib (Gent and Walsh, 1999), a large problem library widely used for benchmarking by the CP community. It is important to note that although the problem domains seem academic, they are important because they appear hidden in many real-world applications. For instance, the quasigroup completion problem is included in scheduling problems.

Our analysis considers the ASP decompositions that we have studied in this chapter. We denote the support encoding of the global constraints by S, the bound encoding of the global constraints by B, and the range encoding of the global constraints by R. To explore the impact of small Hall intervals, we also tried B_k and R_k, a decomposition of the ALL-DIFFERENT constraint with only those cardinality constraint rules (5.2) for which $u - l + 1 \leq k$. The consistency achieved by B_k and R_k is therefore weaker than bound and range consistency, respectively. An implemen-

tation is available as a preliminary version of the CASP system *inca*[2] introduced in the sequel of this thesis.

We also include the hybrid systems *clingcon* (0.1.2)[3], and *ezcsp* (1.6.9)[4] in our empirical analysis. While *clingcon* extends the ASP system *clingo* (2.0.2)[3] with the CP library *gecode* (2.2.0)[5], *ezcsp* combines the grounder *gringo* (2.0.3)[3] and ASP solver *clasp* (1.3.0)[3] with *sicstus* (4.0.8)[6] as the CP solver. Note that *clingo* stands for *clasp on gringo* and combines both systems in a monolithic way. Hence, to provide a representative comparison with *clingcon* and *ezcsp*, we selected the ASP system *clingo* (2.0.3) to solve our decompositions. We also do not separate time spend on grounding/decomposing and solving the problem, because we found the grounder's share of the overall run time is largely insignificant on our benchmark sets.

To compare the performance of CASP solvers against CP systems, we also report results of *gecode* (3.2.0) on the same model, using the *smallest domain* variable selection heuristic. This is also the default in *clingcon*. We also applied the default propagation algorithms in *clingcon*, *ezcsp*, and *gecode*, all of which achieve arc consistency on the binary decomposition of ALL-DIFFERENT.

Experiments were run on a 2.00 GHz PC under Linux. We report results in seconds, where each run was limited to 600 s time and 1 GByte RAM. Run times below 0.1s were always rounded to 0.1s.

Pigeon Hole Problems

The famous *pigeon hole problem* (PHP) is to show that it is not possible to assign n pigeons to $n-1$ holes if each pigeon must be assigned a distinct hole. It is a classic example of a benchmark that aims at the worst-case for many methods that compute ALL-DIFFERENT.

As can be seen from the results shown in Table 5.1, our bound and range encodings perform significantly faster compared to weaker encodings, the hybrid CASP systems, and *gecode*, all of which are limited in their pruning by the propagation they achieve. As can be expected on such problems, detecting large Hall intervals is essential.

[2] `http://potassco.sourceforge.net/labs.html`
[3] `http://potassco.sourceforge.net/`
[4] `http://www.mbal.tk/ezcsp/`
[5] `http://www.gecode.org/`
[6] `http://www.sics.se/sicstus/`

Table 5.1: Runtime results in seconds for PHP.

n	S	B_1	B_2	B_3	B	R_3	R	*ezcsp*	*clingcon*	*gecode*
10	5.4	0.7	0.5	**0.1**	**0.1**	0.2	**0.1**	1.8	1.4	0.9
11	46.5	3.5	1.5	1.0	**0.1**	1.9	**0.1**	16.7	15.2	9.0
12	105.0	14.8	7.1	3.9	**0.1**	2.6	**0.1**	183.9	172.5	104.1
13	—	91.4	68.6	25.4	**0.1**	30.4	**0.1**	—	—	—
14	—	—	350.1	125.0	**0.1**	196.9	**0.1**	—	—	—
15	—	—	—	—	**0.1**	—	**0.1**	—	—	—
16	—	—	—	—	**0.1**	—	**0.1**	—	—	—

Quasigroup Completion

A *quasigroup* is an algebraic structure $(Q, \cdot)$, where Q is a set and $\cdot$ is a binary operation on Q such that for every pair of elements $a, b \in Q$ there exist unique elements $x, y \in Q$ which solve the equations $a \cdot x = b$ and $y \cdot a = b$. The *order* n of a quasigroup is defined by the number of elements in Q. A quasigroup can be represented by an $n \times n$-multiplication table, where for each pair a, b the table gives the result of $a \cdot b$, and it defines a *Latin square*. This means that each element of Q occurs exactly once in each row and each column of the table. The *quasigroup completion problem* (QCP) is to determine whether a partially filled table can be completed in such a way that a multiplication table of a quasigroup is obtained. Randomly generated QCPs have been proposed as a benchmark domain for CP systems by (Gomes and Selman, 1997) because they combine the features of purely random problems and highly structured problems.

In this experiment, we include settings for *gecode* that enforce bound and domain consistency on ALL-DIFFERENT, denoted $gecode^{BC}$ and $gecode^{DC}$, respectively. Table 5.2 gives the run time for solving QCPs of order $n = 20$. The left-most column gives the ratio of preassigned entries.

Our results demonstrate an easy-hard-easy phase transition (Cheeseman et al., 1991) behaviour in the systems *ezcsp*, *clingcon*, *gecode*, and $gecode^{BC}$, while our ASP encodings and $gecode^{DC}$ (not shown) solve all problems within seconds. Employing a propagator that achieves a stronger form of consistency, $gecode^{BC}$ improves over *gecode*. On the other hand, we cannot draw the same conclusion for our approach, i.e., settings S and B. Hence, even our weaker encoding (weaker in terms of consistency) are sufficient to tackle QCPs, and compete with specialised algorithms that aim at a stronger consistency. In other words, such specialised algorithms that enforce domain consistency are not necessary. We attribute the

Table 5.2: Average times over 100 runs on QCP. Timeouts are given in parenthesis.

%	S	B_3	B	R_3	R	*ezcsp*	*clingcon*	*gecode*	*gecode*^{BC}
10	**2.6**	5.0	8.2	6.0	7.3	29.6 (7)	9.7 (4)	2.2 (4)	0.5 (1)
20	**2.4**	5.0	8.0	6.2	7.2	21.3 (20)	6.2 (5)	5.0 (4)	0.9 (3)
30	**2.3**	4.8	7.9	6.1	7.1	10.3 (30)	12.9 (13)	2.9 (13)	1.1 (5)
35	**2.3**	4.8	7.9	6.1	7.0	21.6 (24)	11.2 (17)	14.1 (13)	6.2 (7)
40	**2.3**	4.7	7.8	6.0	6.9	51.6 (29)	23.1 (22)	11.7 (20)	5.7 (9)
45	**2.3**	4.7	7.8	5.9	6.8	36.3 (35)	14.7 (28)	17.7 (25)	6.3 (13)
50	**2.3**	4.6	7.7	5.9	6.8	36.1 (50)	21.2 (37)	25.1 (32)	6.3 (18)
55	**2.3**	4.5	7.6	5.8	6.7	61.4 (51)	24.4 (44)	19.6 (41)	30.9 (29)
60	**2.2**	4.4	7.5	5.6	6.6	60.2 (63)	31.4 (56)	36.0 (51)	27.2 (35)
70	**2.2**	4.2	7.1	5.1	6.0	70.0 (66)	30.2 (50)	28.0 (45)	17.0 (27)
80	**2.1**	4.0	6.7	4.7	5.5	16.2 (18)	4.2 (18)	17.2 (13)	7.0 (7)
90	**2.1**	4.0	6.7	4.7	5.5	1.4	2.6 (1)	0.4 (1)	3.2

advantage of our approach to CDNL which allows for learning interdependencies between the global constraints on rows and columns.

Quasigroup Existence

The *quasigroup existence problem* (QEP) is to determine the existence of certain interesting classes of quasigroups. QEP has been proposed as a benchmark domain for CP systems in (Gent and Walsh, 1999). We follow Fujita et al. (1993) and look at problem classes QG1 to QG7 that were targeted to answer open questions in finite mathematics. Each class gives an axiom that a quasigroup has to satisfy. The axioms are the following:

QG1	If $a \cdot b = c \cdot d$ and $a \cdot b = c \cdot d$ then $a = c$ and $b = d$.
QG2	If $a \cdot b = c \cdot d$ and $a \cdot b = c \cdot d$ then $a = c$ and $b = d$.
QG3	$(a \cdot b) \cdot (b \cdot a) = a$
QG4	$(b \cdot a) \cdot (a \cdot b) = a$
QG5	$((b \cdot a) \cdot b) \cdot b = a$
QG6	$(a \cdot b) \cdot b = a \cdot (a \cdot b)$
QG7	$(b \cdot a) \cdot b = a \cdot (b \cdot a)$

To avoid some symmetries in the search space, we also add the axiom $a \cdot n \geq a - 1$ where n is the order of the desired quasigroup, and assume quasigroups to be *idempotent*, that means $a \cdot a = a$.

All axioms have been modelled in *ezcsp* and *gecode* using constructive disjunction (Van Hentenryck et al., 1995). Their logic programming equivalent are integrity constraints, exploited in the options S, B_k, R_k and *clingcon*.

Our results in Table 5.3 demonstrate that constructive disjunction and integrity constraints have a similar run time behaviour, as for *ezcsp* and *clingcon* on benchmark classes QG1 to QG4. On harder instances, *clingcon*'s way of learning from inconsistent assignments appears to be too costly to be practical. In fact, additional experiments revealed that *clingcon* without learning performs similar to *ezcsp*. On the other hand, our decompositions benefit from learning constraint interdependencies within the CDNL framework, resulting in run times that outperform *clingcon* and *ezcsp*, and compete with *gecode*.

Graceful Graphs

A labelling of the vertices of a graph (V, E) is *graceful* if it assigns a unique label from the integers in $[0, |E|]$ such that, when each edge is labelled with the absolute difference between adjacent vertices, the resulting edge labels are all different. The graceful graph problem (GGP) is to determine the existence of a graceful labelling of a given graph.

We use auxiliary variables for edge labels. Their relation to node labels is represented through a direct encoding, weakening the overall consistency. For our experiments we consider double-wheel graphs that are composed by two copies of a cycle with n vertices, each connected to a central hub. The problem of determining the gracefulness of double-wheel graphs is highly symmetric, similar to PHP, but a solution exists in most cases.

Table 5.4 shows that our encodings compete with *ezcsp* and outperform the other systems, where the support encoding performs better than bound and range encodings. We observe some variability in the results for B_k and R_k, e.g., for $n = 8$ the options B_1 and B solve the problem within the time limit but B_3 does not, although B_3 contains B_1. We explain this variability by the heuristic used in *clingo*, being misled by the extra variables introduced in B_k and R_k. This is inherent with the increasing size of the encoding.

5.2 The GRAMMAR Constraint

One very promising method for scheduling, rostering and sequencing problems is to specify constraints via grammars or automata that recognise some formal lan-

Table 5.3: Runtime results in seconds for QEPs.

	n	S	B_1	B_3	B	R_3	R	*ezcsp*	*clingcon*	*gecode*
	7	1.7	1.7	1.7	1.7	1.7	1.6	65.0	189.8	**0.6**
QG1	8	19.0	5.9	**4.7**	19.8	6.4	**4.7**	—	—	—
	9	—	139.4	152.0	234.6	**27.6**	466.9	—	—	—
	7	1.7	1.7	1.7	1.8	1.7	1.8	46.1	1.5	**1.2**
QG2	8	46.6	**9.6**	10.6	37.7	11.7	14.8	—	—	—
	9	—	246.0	**55.7**	88.3	119.7	213.4	—	—	—
	7	0.2	0.2	0.2	0.3	0.2	0.3	3.2	1.0	**0.1**
QG3	8	0.4	0.4	0.5	0.5	0.5	0.5	4.3	9.0	**0.2**
	9	10.2	**7.4**	9.5	16.5	11.0	12.8	—	—	18.2
	7	0.2	0.2	0.2	0.3	0.3	0.3	2.8	0.7	**0.1**
QG4	8	0.5	0.6	0.7	0.9	0.8	0.7	27.9	36.8	**0.3**
	9	1.3	**1.0**	2.1	3.0	1.1	0.9	442.1	288.8	3.7
	8	0.4	0.4	0.4	0.5	0.4	0.4	6.9	5.3	**0.1**
	9	0.7	0.8	0.8	0.9	0.8	0.8	249.2	—	**0.1**
QG5	10	1.6	1.5	1.6	1.9	1.6	1.6	—	—	**0.2**
	11	2.1	2.2	2.4	3.4	2.8	2.4	—	—	**0.8**
	12	27.0	**6.2**	9.1	12.4	8.4	10.4	—	—	16.4
	8	0.4	0.4	0.5	0.5	0.5	0.4	0.8	—	**0.1**
	9	0.7	0.7	0.8	0.9	0.8	0.8	1.2	—	**0.1**
QG6	10	1.2	1.4	1.5	1.8	1.6	1.5	10.5	—	**0.1**
	11	2.7	2.8	4.0	4.2	3.9	4.8	125.5	—	**1.2**
	12	32.0	**12.9**	25.6	36.4	25.7	50.6	—	—	24.6
	8	0.4	0.4	0.4	0.6	0.5	0.5	1.1	—	**0.1**
QG7	9	**0.7**	1.0	1.2	1.7	1.2	1.4	9.1	—	0.9
	10	6.7	**3.2**	5.2	8.0	4.7	4.6	—	—	22.0

Table 5.4: Runtime results in seconds for GGP.

n	S	B_1	B_3	B	R_3	R	*ezcsp*	*clingcon*	*gecode*
3	11.4	3.8	5.7	8.7	6.0	10.4	6.5	66.9	**1.8**
4	1.3	2.0	1.5	3.2	3.0	2.5	0.6	**0.1**	**0.1**
5	4.5	5.0	4.5	13.5	12.5	31.4	1.0	2.0	**0.1**
6	7.2	11.0	17.6	47.7	21.3	110.2	**1.2**	—	7.2
7	23.8	28.3	67.9	227.9	60.0	432.9	**18.0**	—	—
8	48.4	68.4	—	207.8	58.4	356.8	**4.3**	—	—
9	**82.8**	106.5	200.4	486.6	227.4	—	390.5	—	—

guage (Sellmann, 2006; Quimper and Walsh, 2006). For instance, we might want to ensure that *anyone working three consecutive shifts then has two or more days off*, or that *an employee changes activities only after a fifteen minutes break or one hour lunch*. Intuitively, GRAMMAR constraints require that the sequence of values taken by the variables in their scope belong to a formal language produced by a CFG.

Definition 5.4: GRAMMAR

Given a CFG $\mathscr{G}$ and an assignment A, the constraint GRAMMAR($\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) is satisfied if and only if $A(\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle) \in L_{\mathscr{G}}$.

Grammar-based constraint propagators were proposed by Sellmann (2006) and Quimper and Walsh (2006), and modelled with SAT by Quimper and Walsh (2007). Although SAT models can be directly converted into ASP (Niemelä, 1999), we here show that GRAMMAR and related constraints can be modelled with ASP in a more straightforward and easily maintainable way without paying any penalty, e.g., in form of efficiency, for using ASP. Moreover, the propagation of an ASP solver can achieve domain consistency on the original constraint. We make several contributions to this line of research.

- First, we start with modelling GRAMMAR constraints that are specified using CFG with ASP. We will show that they can be made to not hinder propagation, i.e., the inference of an ASP solver can prune all possible values. We also provide results on asymptotic run time complexity. For restricted classes of CFGs, for instance, linear or regular grammars, we will adapt our encodings and report improved run times.

- Second, we present and study alternative ASP encodings for a special case of GRAMMAR constraints, i.e., REGULAR constraints that are specified via DFAs. Again, we give theoretical results on the pruning achieved by the UP inference of any ASP solver, and run time complexity.

- Third, we demonstrate the applicability of our approach on shift-scheduling instances.

Context-free Grammars in Chomsky Normal Form

We here encode GRAMMAR($\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) with ASP in a very straightforward way, based on the well known Cocke-Younger-Kasami (CYK; Younger, 1967) parser. Our

construction will maintain a CYK parsing table T where $A \in T[i,j]$ if A is a non-terminal symbol that produces the string $t_i t_{i+1} \dots t_{i+j} \in \Sigma^*$ and the assignments $\mathsf{v}_{i+\ell} = t_{i+\ell}$ are possible, i.e., the value $t_{i+\ell}$ has not been removed from the domain of $\mathsf{v}_{i+\ell}$, where $0 \leq \ell \leq j$.

Definition 5.5: ASP Encoding of GRAMMAR Constraints

Our ASP encoding of the GRAMMAR($\mathscr{G}, \langle \mathsf{v}_1, \dots, \mathsf{v}_n \rangle$) constraint is constructed as follows:

- To represent $A \in T[i,j]$ whether A produces the string of length j starting from the i-th symbol, we introduce the new atom $A(i,j)$.
- Each production of the form $A ::= t$ is encoded by rules of the form

$$A(i,1) \leftarrow [[\mathsf{v}_i = t]] \tag{5.5}$$

  where $1 \leq i \leq n$. A rule of the form (5.5) states that $A \in T[i,1]$ if $\mathsf{v}_i = t$, i.e., A produces the i-th symbol t.
- Each production of the form $A ::= BC$ is encoded by rules of the form

$$A(i,j) \leftarrow B(i,k), C(i+k, j-k) \tag{5.6}$$

  where $1 \leq i < n$ and $1 \leq k < j \leq n-i-1$. Intuitively, a rule of the form (5.6) states that A produces the string of j symbols starting at the i-th symbol if B produces the string of length k starting from the i-th symbol and C produces the string of length $j-k$ starting from the $i+k$-th symbol. In other words, k splits the string produces by A into the (sub-)strings produced by $B \in T[i,k]$ and $C \in T[i+k, j-k]$.
- Finally, the constraint is violated if the dedicated start symbol S does not produce the input string, captured by the rule

$$[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \dots, \mathsf{v}_n \rangle)]] \leftarrow not\ S(1,n). \tag{5.7}$$

We denote the encoding by ASP-GRAMMAR[$\mathscr{G}, \langle \mathsf{v}_1, \dots, \mathsf{v}_n \rangle$].

Observe that our ASP encoding of the GRAMMAR constraint is not only straightforward, but easily maintainable, as it amounts to a syntactic transformation of the production rules in the grammar.

The asymptotic space complexity of $\text{ASP-GRAMMAR}[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$ is $\mathscr{O}(n^3|\mathscr{G}|)$. We can prove that our encoding is correct.

Theorem 5.7

$\text{GRAMMAR}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)$ is satisfiable if and only if there exists a solution $\mathbf{A}$ for $\Delta_{\text{ASP-GRAMMAR}[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle] \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]}$ s.t. $[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]] \in \mathbf{A}^{\mathbf{F}}$.

Proof. Correctness follows from the correctness of the CYK parser. To improve readability, let $\Psi = \text{ASP-GRAMMAR}[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle] \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$. We show both implications of the theorem.

($\Rightarrow$) Suppose $\text{GRAMMAR}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)$ is satisfiable, i.e., there exists an assignment to the sequence of variables $\omega = \mathsf{v}_1 \ldots \mathsf{v}_n$, say ω is assigned the string $t_1 \ldots t_n$, such that the starting nonterminal S belongs to $T[1, n]$. We can construct an assignment $\mathbf{A}$ the following way:

- $\mathbf{T}[[\mathsf{v}_i = t_i]] \in \mathbf{A}$ where $1 \leq i \leq n$, encoding the satisfying assignment to the sequence of variables
- $\mathbf{T}A(i, j) \in \mathbf{A}$ for all nonterminals $A \in T[i, j]$, i.e., those that appear in the CYK table, where $1 \leq i \leq n$ and $1 \leq j \leq n - i + 1$,
- all bodies are set to the value equivalent to the conjunction of their elements, and all other atoms are set to false.

We observe that no nogood in Δ_Ψ is contained in $\mathbf{A}$, i.e., $\mathbf{A}$ is conflict-free. Since $\mathbf{A}$ is also total, $\mathbf{A}$ is a solution for Δ_Ψ.

($\Leftarrow$) Let $\mathbf{A}$ be a solution for the set Δ_Ψ such that $[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]] \in \mathbf{A}^{\mathbf{F}}$. We show that $\text{GRAMMAR}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)$ is satisfiable.

To begin, we prove that for any nonterminal A that $A(i, j) \in \mathbf{A}^{\mathbf{T}}$ implies $A \Rightarrow^*_{\mathscr{G}} t_i \ldots t_{i+j-1}$ by structural induction.

Without loss of generality, suppose the production rules in $\mathscr{G}$ with A on the right-hand-side be

$$A ::= B_1 C_1 \mid \cdots \mid B_m C_m \mid t_{i_{m+1}} \mid \cdots \mid t_{i_\ell}.$$

The base case is $A(i, 1) \in \mathbf{A}^{\mathbf{T}}$. For $A(i, 1)$ we have the following rules of the form (5.5)

$$\begin{aligned} r_{m+1}: &\quad A(i,1) \leftarrow [[\mathsf{v}_i = t_{i_{m+1}}]] \\ &\quad \ldots \\ r_\ell: &\quad A(i,1) \leftarrow [[\mathsf{v}_i = t_{i_\ell}]] \end{aligned}$$

in ASP-GRAMMAR$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$. Hence, there is a nogood

$$\{\mathbf{F}\mathrm{body}(r_m + 1), \ldots, \mathbf{F}\mathrm{body}(r_\ell), \mathbf{T}A(i, j)\} \in \mathbf{\Delta}_\Psi^{A(i,1)} \subseteq \Delta_\Psi$$

guaranteeing that at least one production rule applies, i.e., $\mathbf{T}\mathrm{body}(r_s)$ for some s with $m < s \leq \ell$ is in $\mathbf{A}$. Then, the nogood

$$\{\mathbf{T}\mathrm{body}(r_s), \mathbf{F}[[\mathsf{v}_i = t_{i_s}]]\} \in \mathrm{EQ}_{\mathrm{body}(r_s)} \subseteq \Delta_\Psi$$

guarantees $[[\mathsf{v}_i = t_{i_s}]] \in \mathbf{A}^{\mathbf{T}}$, i.e., we have the assignment $\mathsf{v}_i = t_{i_s}$. Moreover, the nogoods represented by ASP-VALUE$[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$ guarantee that for each index i, where $1 \leq i \leq n$, we have $[[\mathsf{v} = t_i]]$ for precisely one terminal t_i, i.e., every variable is assigned a terminal symbol. In fact, since $\mathbf{A}$ is conflict-free, we have the equality $t_i = t_{i_s}$. Hence, $A \Rightarrow_{\mathscr{G}}^* t_i$.

We now show for any nonterminal A such that $A(i, j) \in \mathbf{A}^{\mathbf{T}}$ and $j > 1$, assuming that $A'(i, j') \in \mathbf{A}^{\mathbf{T}}$ implies $A' \Rightarrow_{\mathscr{G}}^* t_i \ldots t_{i+j'-1}$ holds for every nonterminal A' where $1 \leq j' < j$. To begin, for parsings of length j starting from the i-th symbol, we have the following rules of the form (5.5–5.6)

$$\begin{aligned} r_1 :&\quad A(i, j) \leftarrow B_1(i, k),\ C_1(i + k, j - k) \\ &\ \ldots \\ r_m :&\quad A(i, j) \leftarrow B_m(i, k),\ C_m(i + k, j - k) \end{aligned}$$

in ASP-GRAMMAR$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$, where $1 \leq k < j$. Hence, there is a nogood

$$\{\mathbf{F}\mathrm{body}(r_1), \ldots, \mathbf{F}\mathrm{body}(r_\ell), \mathbf{T}A(i, j)\} \in \mathbf{\Delta}_\Psi^{A(i,j)} \subseteq \Delta_\Psi$$

guaranteeing that at least one production rule applies, i.e., $\mathbf{T}\mathrm{body}(r_s)$ for some s with $1 \leq s \leq m$. Then, the nogoods

$$\{\mathbf{T}\mathrm{body}(r_s), \mathbf{F}B_s(i, k)\}, \{\mathbf{T}\mathrm{body}(r_s), \mathbf{F}C_s(i + k, j - k)\} \in \mathrm{EQ}_{\mathrm{body}(r_s)} \subseteq \Delta_\Psi$$

guarantee $B_s(i, k), C_s(i + k, j - k) \in \mathbf{A}^{\mathbf{T}}$. By assumption, we have B_s produces the string from the i-th to the $i + k - 1$-th symbols, and C_s produces the string from the $i + k$-th to the $i + j - 1$-th symbols, i.e., $B_s \Rightarrow_{\mathscr{G}}^* t_i \ldots t_{i+k-1}$ and $C_s \Rightarrow_{\mathscr{G}}^* t_{i+k} \ldots t_{j-k}$. Therefore, we have $A \Rightarrow_{\mathscr{G}}^* t_i \ldots t_{i+j-1}$.

We conclude that $A(i, j) \in \mathbf{A}^{\mathbf{T}}$ implies $A \Rightarrow_{\mathscr{G}}^* t_i \ldots t_{i+j-1}$ for any nonterminal A where $1 \leq i \leq n$ and $1 \leq j < n - i + 1$. Now, the nogoods represented by the rule r of the form (5.7), i.e., the nogoods $\{\mathbf{F}[[\overline{\mathrm{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]], \mathbf{T}\mathrm{body}(r)\}$ and $\{\mathbf{F}\mathrm{body}(r), \mathbf{F}S(1, n)\}$, guarantee that $S(1, n) \in \mathbf{A}^{\mathbf{T}}$. Hence, $S \Rightarrow_{\mathscr{G}}^* t_1 \ldots t_n$, i.e., the constraint GRAMMAR$(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)$ is satisfied. □

Unfortunately, our straightforward encoding is not very efficient from a theoretical point of view, i.e., it does not prune all possible values.

Counter Example 5.4

Reconsider the grammar from Example 2.5, i.e., the CFG $\mathcal{G}$ given through the productions

$$S ::= SA \mid AS \mid 2$$
$$A ::= 1$$

Suppose the input string of length 2, $\mathsf{v}_1\mathsf{v}_2$ with $\mathrm{dom}(\mathsf{v}_1) = \mathrm{dom}(\mathsf{v}_2) = [1,3]$. Our encoding ASP-GRAMMAR$[\mathcal{G}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle]$ comprises the following rules

$$A(1,1) \leftarrow [[\mathsf{v}_1 = 1]] \qquad S(1,1) \leftarrow [[\mathsf{v}_1 = 2]]$$
$$A(2,1) \leftarrow [[\mathsf{v}_2 = 1]] \qquad S(2,1) \leftarrow [[\mathsf{v}_2 = 2]]$$
$$S(1,2) \leftarrow S(1,1),\ A(2,1)$$
$$S(1,2) \leftarrow A(1,1),\ S(2,1)$$
$$[[\overline{\text{GRAMMAR}}(\mathcal{G}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle)]] \leftarrow not\ S(1,2)$$

Although the value 3 does not appear in any assignment that satisfies the constraint GRAMMAR$(\mathcal{G}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle)$, applying UP on ASP-GRAMMAR$[\mathcal{G}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle]$ does not prune 3 from the domains.

A SAT model of the GRAMMAR constraint such that UP prunes all values was proposed by Quimper and Walsh (2007). The encoding is based on their decomposition into an and-or-graph (Quimper and Walsh, 2006).

To achieve a similar result, we propose an extension to our ASP encoding in which we take into account whether an entry in the CYK table acts in a successful parsing of the input string.

Definition 5.6: Domain Consistent ASP Encoding of GRAMMAR Constraints

Our improved ASP encoding of the GRAMMAR$(\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)$ constraint is constructed as follows:

- Include ASP-GRAMMAR$[\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$.
- We introduce additional atoms $\mathbf{act}A(i,j)$ and $\mathbf{act}BC(i,j)$ to represent whether $S \Rightarrow^*_{\mathcal{G}} v_1 A v_2 \Rightarrow^*_{\mathcal{G}} \omega$ and $S \Rightarrow^*_{\mathcal{G}} v_1 BC v_2 \Rightarrow^*_{\mathcal{G}} \omega$ with $\omega \in L_{\mathcal{G}}$, re-

spectively, i.e., A or BC, respectively, act in a successful parsing of the string of length j starting from the i-th symbol.

– For each production of the form $A ::= t$ we encode rules of the form

$$\{\mathbf{act}A(i,1)\} \leftarrow [[\mathsf{v}_i = t]] \tag{5.8}$$

where $1 \leq i \leq n$. A rule of the form (5.8) states that A produces the i-th symbol if $\mathsf{v}_i = t$, and may act in a successful parsing of the input string.

– Similarly, for each production of the form $A ::= BC$ we encode the following two rules, using the auxiliary atom $\mathbf{act}BC(i,j)$,

$$\{\mathbf{act}A(i,j)\} \leftarrow \mathbf{act}BC(i,j) \tag{5.9}$$

$$\{\mathbf{act}BC(i,j)\} \leftarrow \mathbf{act}B(i,k),\ \mathbf{act}C(i+k,j-k) \tag{5.10}$$

where $1 \leq i < n$ and $1 \leq k < j \leq n-i-1$. Intuitively, rules of the form (5.9) encode that $A \in T[i,j]$ may act in a successful parsing if A produces the pair BC and BC acts in a successful parsing, producing the string of length j starting from the i-th symbol.

In turn, rules of the form (5.10) encode that the pair BC produces the string of length j from the i-th symbol and may act in a successful parsing if $B \in T[i,k]$ and $C \in T[i+k, j-k]$ act in a successful parsing of the input string. Similar to rules of the form (5.6), k splits the string produced by BC into the (sub-)strings produced by B and C.

– To ensure that UP prunes all possible values, we capture the condition that for each assignment $\mathsf{v}_i = t$ there exist a nonterminal that produces t and acts in a successful parsing of the input string, forming a *support* for $\mathsf{v}_i = t$. Let $A_1, \ldots, A_m \in N$ such that $A_\ell ::= t$, for all $1 \leq \ell \leq m$. We include

$$\leftarrow [[\mathsf{v}_i = t]],\ not\ \mathbf{act}A_1(i,1),\ \ldots,\ not\ \mathbf{act}A_m(i,1) \tag{5.11}$$

into our encoding. Similarly, for each pair BC that produces the string of j symbols starting at the i-th symbol and acts in a successful parsing of the input string, there must be a nonterminal that produces BC and acts in a successful parsing of the input string. Let $A_1, \ldots, A_m \in N$ such

that $A_\ell ::= BC$, for all $1 \leq \ell \leq m$, we encode this condition by integrity constraints of the form

$$\leftarrow \mathbf{act}BC(i,j),\ not\ \mathbf{act}A_1(i,j),\ \ldots,\ not\ \mathbf{act}A_m(i,j) \tag{5.12}$$

where $1 \leq i < n$ and $1 \leq j \leq n-i$.

In turn, for each A that produces the string of j symbols starting at the i-th symbol and acts in a successful parsing of the input string, there must be a pair of nonterminals containing A on the right-hand side of some production rule, i.e., AB or BA for some nonterminal B, that act in a successful parsing of the input string. Let $AB_1, \ldots, AB_k$ and $B_{k+1}A, \ldots, B_mA$ all such pairs of nonterminals from the production rules of $\mathscr{G}$. We encode this condition by integrity constraints of the form

$$\begin{aligned} \leftarrow \mathbf{act}A(i,j),\ & not\ \mathbf{act}AB_1(i,j+\ell_1),\ \ldots, \\ & not\ \mathbf{act}AB_k(i,j+\ell_k), \\ & not\ \mathbf{act}B_{k+1}A(i-\ell_{k+1}, j+\ell_{k+1}),\ \ldots, \\ & not\ \mathbf{act}B_mA(i-\ell_m, j+\ell_m) \end{aligned} \tag{5.13}$$

where $1 \leq i < n$, $1 \leq j \leq n-i$, $1 \leq \ell_h \leq n-i-j$ for $1 \leq h \leq k$, and $1 \leq \ell_h \leq i-1$ for $k < h \leq m$.

– The condition that the unique starting symbol S has to produce the input string remains unchanged. In fact, S must act in a succesful parsing, or otherwise the original constraint is violated.

$$\leftarrow not\ \mathbf{act}S(1,n) \tag{5.14}$$

We denote the improved encoding by ASP-GRAMMAR-DC$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$.

It is important to note that above encoding assumes $\mathbf{F}[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]]$, i.e., the constraint $[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]]$ shall not be violated, but it can be easily adjusted to work with any assignment to $[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]]$ by carefully including the default negated atom not $[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]]$ into the integrity constraints that encode support.

The space complexity of ASP-GRAMMAR-DC$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$, in comparison to its simpler counterpart, remains unchanged. It is still $\mathscr{O}(n^3|\mathscr{G}|)$. Hence, there is no penalty in terms of asymptotic size of the encoding. Though, there are $\mathscr{O}(n^2|N|)$ rules of the form (5.13) that have a rather large body, i.e., with an asymptotic space

complexity of $\mathcal{O}(n|\{\omega \mid N ::= \omega \in \mathcal{G}\}|)$.

Example 5.5

Reconsider the CFG $\mathcal{G}$ from Counter Example 5.4, again applied to the string $\mathsf{v}_1\mathsf{v}_2$ with $\text{dom}(\mathsf{v}_1) = \text{dom}(\mathsf{v}_2) = [1,3]$. In addition to the ones specified in Counter Example 5.4, our ASP encoding ASP-GRAMMAR-DC$[\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$ comprises the following rules:

$$\{\mathbf{act}A(1,1)\} \leftarrow [[\mathsf{v}_1 = 1]] \qquad \{\mathbf{act}S(1,1)\} \leftarrow [[\mathsf{v}_1 = 2]]$$

$$\{\mathbf{act}A(2,1)\} \leftarrow [[\mathsf{v}_2 = 1]] \qquad \{\mathbf{act}S(2,1)\} \leftarrow [[\mathsf{v}_2 = 2]]$$

$$\{\mathbf{act}S(1,2)\} \leftarrow \mathbf{act}AS(1,2)$$

$$\{\mathbf{act}AS(1,2)\} \leftarrow \mathbf{act}A(1,1),\ \mathbf{act}S(2,1)$$

$$\{\mathbf{act}S(1,2)\} \leftarrow \mathbf{act}SA(1,2)$$

$$\{\mathbf{act}SA(1,2)\} \leftarrow \mathbf{act}S(1,1),\ \mathbf{act}A(2,1)$$

$$\leftarrow [[\mathsf{v}_1 = 1]],\ not\ \mathbf{act}A(1,1) \qquad \leftarrow \mathbf{act}A(1,1),\ not\ \mathbf{act}AS(1,2)$$

$$\leftarrow [[\mathsf{v}_1 = 2]],\ not\ \mathbf{act}S(1,1) \qquad \leftarrow \mathbf{act}A(2,1),\ not\ \mathbf{act}SA(1,2)$$

$$\leftarrow [[\mathsf{v}_1 = 3]] \qquad \leftarrow \mathbf{act}S(1,1),\ not\ \mathbf{act}SA(1,2)$$

$$\leftarrow [[\mathsf{v}_2 = 1]],\ not\ \mathbf{act}A(2,1) \qquad \leftarrow \mathbf{act}S(2,1),\ not\ \mathbf{act}AS(1,2)$$

$$\leftarrow [[\mathsf{v}_2 = 2]],\ not\ \mathbf{act}S(2,1) \qquad \leftarrow \mathbf{act}SA(1,2),\ not\ \mathbf{act}S(1,2)$$

$$\leftarrow [[\mathsf{v}_2 = 3]] \qquad \leftarrow \mathbf{act}AS(1,2),\ not\ \mathbf{act}S(1,2)$$

$$\leftarrow not\ \mathbf{act}S(1,2)$$

Running UP on ASP-GRAMMAR-DC$[\mathcal{G}, \langle \mathsf{v}_1, \mathsf{v}_2 \rangle] \cup$ ASP-VALUE$[\{\mathsf{v}_1, \mathsf{v}_2\}]$ prunes the value 3 from the domains, i.e., sets $[[\mathsf{v}_1 = 3]]$ and $[[\mathsf{v}_2 = 3]]$ to false.

With our improved encoding, UP provides an efficient domain consistency propagator for free.

Theorem 5.8: Domain Consistency on GRAMMAR via Unit Propagation

UP on ASP-GRAMMAR-DC$[\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle] \cup$ ASP-VALUE$[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$ enforces domain consistency on GRAMMAR$(\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)$.

Proof. The proof strategy is to show that if an assignment to a variable is possible then there is a combination of values to the other variables that satisfies the constraint.

Let **A** be an assignment to the atoms in a value representation of the variables in the scope of the constraint, i.e., the atoms in the program ASP-VALUE$[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$,

representing the current set of values in the domain of the variables. Suppose UP has been run to completion, extending $\mathbf{A} \cup \{\mathbf{F}[[\overline{\text{GRAMMAR}}(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]]\}$ (i.e., the GRAMMAR constraint shall not be violated) to the assignment $\mathbf{A}'$.

Consider any variable v_i in the scope of the constraint and a value t that has not been removed from the domain of v_i. That means, $[[\mathsf{v}_i = t]] \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., $[[\mathsf{v}_i = t]]$ is either true or unassigned. Then, the nogood $\{\mathbf{T}[[\mathsf{v}_i = t]]\} \cup \{\mathbf{F}\ \mathbf{act}A(i,1) \mid A ::= t \in \mathscr{G}\}$ represented by (5.11) guarantees that there is a nonterminal A which produces t as the i-th symbol, i.e., $\mathbf{act}A(i,1) \notin (\mathbf{A}')^{\mathbf{F}}$.

Now, consider any nonterminal $A \in N$ such that $A(i,j) \notin (\mathbf{A}')^{\mathbf{F}}$ where $1 \leq i \leq n$ and $1 \leq j \leq n-i+1$. The nogood $\{\mathbf{T}A(i,j), \mathbf{F}\ \mathbf{act}AB_1(i, j+\ell_1), \ldots, \mathbf{F}\ \mathbf{act}AB_k(i, j+\ell_k), \mathbf{F}\ \mathbf{act}B_{k+1}A(i-\ell_{k+1}, j+\ell_{k+1}), \ldots, \mathbf{F}\ \mathbf{act}B_mA(i-\ell_m, j+\ell_m\}$ represented by (5.13) guarantees that there must be a pair of nonterminals AB or BA (i.e., containing A), that can be produced, and the corresponding atom is not in $(\mathbf{A}')^{\mathbf{F}}$. In turn, for any pair of nonterminals BC such that $\mathbf{act}BC(i,j) \notin (\mathbf{A}')^{\mathbf{F}}$ where $1 \leq i \leq n$ and $1 \leq j \leq n-i+1$, the nogood $\{\mathbf{T}\ \mathbf{act}BC(i,j), \mathbf{F}\ \mathbf{act}A_1(i,j), \ldots, \mathbf{F}\ \mathbf{act}A_m(i,j)\}$ represented by (5.12) guarantees that there is some $\mathbf{act}A_k(i,j) \notin (\mathbf{A}')^{\mathbf{F}}$ for some $1 \leq k \leq m$ such that $A_k ::= BC \in \mathscr{G}$.

By successively applying these arguments, we obtain a sequence of productions connecting $[[\mathsf{v}_i = t]]$ to $S(1,n)$, or rather, a sequence connecting $S(1,n)$ to $[[\mathsf{v}_i = t]]$ representing a successful parsing of an input string, such that every atom in this sequence is not in $(\mathbf{A}')^{\mathbf{F}}$. In fact, for any atom in this sequence, nogoods represented by the remaining rules (5.8–5.10) and (5.14) guarantee that the corresponding nonterminal produces a string of values $t_1 \ldots t \ldots t_n \in L_{\mathscr{G}}$ that are in the domains of the corresponding variables. (The proof repeats the arguments from the proof of Theorem 5.7.) In other words, we have $[[\mathsf{v}_1 = t_1]], \ldots, [[\mathsf{v}_i = t]], \ldots, [[\mathsf{v}_n = t_n]] \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., the value t_ℓ is in the domain of v_ℓ for $1 \leq \ell \leq n$ with $\ell \neq i$.

In conclusion, when a variable v_i is assigned a value t one can find compatible values in the domains of all the other variables. Since at least one value must be in each domain, guaranteed by (4.2), we have a set of non-empty domains which are domain consistent. □

Now we address the run time. Note that the specialised algorithm of Sellmann (2006) and the approach of Quimper and Walsh (2007) have the same cubic asymptotic run time complexity, i.e., the worst case run time complexity of the CYK algorithm.

Theorem 5.9

UP on ASP-GRAMMAR-DC[$\mathcal{G}$, $\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$] $\cup$ ASP-VALUE[$\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$] runs in $\mathcal{O}(n^3|\mathcal{G}|)$ time down any branch of the search tree.

Proof. For each of the n variables, there are $\mathcal{O}(d)$ constant-size nogoods and one nogood of size $\mathcal{O}(d)$ represented by ASP-VALUE[$\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$]. On the other hand, there are $\mathcal{O}(|\mathcal{G}|n^3)$ nogoods of constant size and $\mathcal{O}(n^2|N|)$ nogoods of $\mathcal{O}(n|\{\omega \mid N ::= \omega \in \mathcal{G}\}|)$ size represented by ASP-GRAMMAR-DC[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$]. By Theorem 2.1 we obtain, down any branch of the search tree, a total running time complexity given by $\mathcal{O}(nd) + \mathcal{O}(n^3|\mathcal{G}|) = \mathcal{O}(n^3|\mathcal{G}|)$. □

Context-free Grammars with Restrictions

An extension which is sometimes useful in practice but goes slightly beyond CFGs considers restrictions on productions (Quimper and Walsh, 2007). For instance, in the shift-scheduling domain (see Experimental Results), we want that *an employee works on an activity for a minimum of one hour*. We will assume that such restrictions are represented via (external) atoms $\mathsf{f}_A(i, j)$ limiting the start i and length j of a string produced by a nonterminal symbol A.

Definition 5.7: ASP Encoding of Restricted GRAMMAR Constraints

We amend ASP-GRAMMAR[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$] by encoding restrictions on a production of the form $A ::= BC$, with restrictions represented by $\mathsf{f}_A(i, j)$,$\mathsf{f}_B(i, j)$, and $\mathsf{f}_C(i, j)$, by rules of the form

$$A(i, j) \leftarrow B(i, k),\ C(i + k, j - k),\ \mathsf{f}_A(i, j),\ \mathsf{f}_B(i, k),\ \mathsf{f}_C(i + k, j - k) \qquad (5.15)$$

where $1 \leq i \leq n$, $1 \leq j \leq n - i + 1$. A rule of the form (5.15) encodes that the nonterminal A produces a string of length j starting at the i-th symbol if

- the condition $\mathsf{f}_A(i, j)$ is satisfied,
- B produces a string of length k starting at the i-th symbol where the condition $\mathsf{f}_B(i, k)$ is satisfied, and
- C produces a string of length $j - k$ starting at the $i + k$-th symbol where the condition $\mathsf{f}_C(i + k, j - k)$ is satisfied.

The changes to ASP-GRAMMAR-DC$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$ are symmetric, i.e., we use rules of the form

$$\{\mathbf{act}A(i,j)\} \leftarrow \mathbf{act}BC(i,j),\ \mathsf{f}_A(i,j) \tag{5.16}$$

$$\begin{aligned}\{\mathbf{act}BC(i,j)\} \leftarrow\ & \mathbf{act}B(i,k),\ \mathbf{act}C(i+k, j-k), \\ & \mathsf{f}_B(i,k),\ \mathsf{f}_C(i+k, j-k)\end{aligned} \tag{5.17}$$

where $1 \leq i < n$ and $1 \leq k < j \leq n-i-1$.

Amendments to restrictions on productions of the form $A ::= t$ are similar. For ASP-GRAMMAR$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$ we use rules of the form

$$A(i,1) \leftarrow [[\mathsf{v}_i = t]], \mathsf{f}_A(i,1) \tag{5.18}$$

where $1 \leq i \leq n$, $1 \leq j \leq n-i+1$. This rule states that A produces the i-th symbol if $\mathsf{v}_i = t$, i.e., the i-th symbol is the terminal t, and the condition $\mathsf{f}_A(i,1)$ is satisfied. Our encoding ASP-GRAMMAR-DC$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$ is amended by rules of the form

$$\{\mathbf{act}A(i,1)\} \leftarrow [[\mathsf{v}_i = t]],\ \mathsf{f}_A(i,1) \tag{5.19}$$

where $1 \leq i < n$ and $1 \leq k < j \leq n-i-1$.

Linear Grammars

In some cases, we only need a linear language to specify problem constraints. They are a special case of CFG where all productions are of the form $A ::= tB$, $A ::= Bt$. To also capture productions of these forms, we make few simple changes to our encodings.

Definition 5.8: ASP Encoding of Linear GRAMMAR Constraints

Following our previous encodings we represent production rules of the form $A ::= tB$ and $A ::= Bt$ by rules of the form

$$A(i,j) \leftarrow [[\mathsf{v}_i = t]],\ B(i+1, j-1) \tag{5.20}$$

$$A(i,j) \leftarrow B(i, j-1),\ [[\mathsf{v}_{i+j-1} = t]] \tag{5.21}$$

in ASP-GRAMMAR[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$], and

$$\{\mathbf{act}A(i,j)\} \leftarrow [[\mathsf{v}_i = t]],\ \mathbf{act}B(i+1, j-1) \tag{5.22}$$

$$\{\mathbf{act}A(i,j)\} \leftarrow \mathbf{act}B(i, j-1),\ [[\mathsf{v}_{i+j-1} = t]] \tag{5.23}$$

in ASP-GRAMMAR-DC[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$], where $1 \leq i \leq n$, $1 \leq j \leq n-i+1$.

For linear languages, propagation is faster.

Theorem 5.10

If $\mathcal{G}$ is linear then UP on

ASP-GRAMMAR-DC[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$] $\cup$ ASP-VALUE[$\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$]

enforces domain consistency on GRAMMAR($\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) in $\mathcal{O}(n^2|\mathcal{G}|)$ time down any branch of the search tree.

Proof. The proof follows the one of Theorem 5.9.

For each of the n variables, there are $\mathcal{O}(d)$ constant-size nogoods and one nogood of size $\mathcal{O}(d)$ represented by ASP-VALUE[$\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$]. On the other hand, if all productions are of the form $A ::= tB$, $A ::= Bt$, or $A ::= t$, then there are $\mathcal{O}(n^2|\mathcal{G}|)$ nogoods of constant size represented by ASP-GRAMMAR-DC[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$]. By Theorems 2.1 we obtain, down any branch of the search tree, a total running time complexity given by $\mathcal{O}(nd) + \mathcal{O}(n^2|\mathcal{G}|) = \mathcal{O}(n^2|\mathcal{G}|)$. □

Regular Grammars

In some cases, we only need a regular language produced by a regular grammar in order to specify problem constraints. In regular grammars, all productions are of the form $A ::= tB$ or $A ::= t$. Hence, a distinctive feature of regular grammars is that each nonterminal $A \in T[i,j]$ produces a string from the i-th symbol to the n-th symbol (the last symbol), i.e., $j = n$ always holds. Using this insight, we can optimise our encodings.

Definition 5.9: ASP Encoding of Regular GRAMMAR Constraints

If $\mathcal{G}$ is regular, following our previous encodings, we encode production rules of the form $A ::= tB$ by rules of the form

$$A(i, n-i+1) \leftarrow [[\mathsf{v}_i = t]],\ B(i+1, n-i) \tag{5.24}$$

where $1 \leq i \leq n$. The changes to ASP-GRAMMAR-DC$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$ are symmetric, i.e., we encode those productions by rules of the form

$$\{\mathbf{act}A(i, n-i+1)\} \leftarrow [[\mathsf{v}_i = t]],\ \mathbf{act}B(i+1, n-i) \tag{5.25}$$

where $1 \leq i \leq n$.

For regular languages, propagation is faster.

Theorem 5.11

If $\mathscr{G}$ is regular then UP on

$$\text{ASP-GRAMMAR-DC}[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle] \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$$

enforces domain consistency on GRAMMAR$(\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)$ in $\mathscr{O}(nd|\mathscr{G}|)$ time down any branch of the search tree.

Proof. The proof follows the one of Theorem 5.9.

For each of the n variables, there are $\mathscr{O}(d)$ constant-size nogoods and one nogood of size $\mathscr{O}(d)$ represented by ASP-VALUE$[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$. On the other hand, if all productions are of the form $A ::= tB$ or $A ::= t$, then there are $\mathscr{O}(n|\mathscr{G}|)$ nogoods of constant size represented by ASP-GRAMMAR-DC$[\mathscr{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$. By Theorems 2.1 we obtain, down any branch of the search tree, a total running time complexity given by $\mathscr{O}(nd) + \mathscr{O}(n|\mathscr{G}|) \leq \mathscr{O}(nd|\mathscr{G}|)$. □

The REGULAR Constraint

Recall that each regular language can also be specified by means of a DFA $\mathscr{M}$ that accepts assignments to a sequence of variables if and only if it is a member of the language. Hence, as an alternative to constraints specified via regular grammars, we present and study encodings of REGULAR constraints (Pesant, 2004) that are specified via DFAs.

An automaton-based constraint propagator was previously modelled with SAT by Quimper and Walsh (2007) and Bacchus (2007). We here follow their idea and represent the constraint with ASP by encoding the processing of $\mathscr{M}$ for a fixed input length. Our encodings are not a direct copy of theirs. As we shall see, our first encoding will capture all of $\mathscr{M}$'s processing with rules of a single form. This makes it more straightforward and easily maintainable. To achieve an encoding that pays no penalty in terms of pruning, however, we will follow Quimper and

Walsh (2007) and Bacchus (2007) by introducing auxiliary atoms in an extended encoding of REGULAR.

We start with a definition of the constraint.

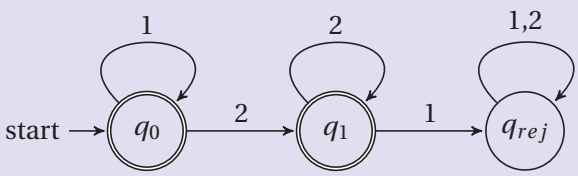
Definition 5.10: REGULAR

Given a DFA $\mathcal{M}$, the REGULAR constraint REGULAR($\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) is satisfied on just those assignments to the sequence of variables $\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$ which belong to the language recognised by $\mathcal{M}$.

The automaton from Example 2.8 will serve as a running example for the remainder of this section.

Example 5.6

Consider the DFA $\mathcal{M} = (\{q_0, q_1, q_{rej}\}, \{1,2\}, \delta, q_0, \{q_0, q_1\})$ where the transition function δ is represented by the following automaton diagram:



Consider the input string of length 3, $\mathsf{v}_1\mathsf{v}_2\mathsf{v}_3$ with the following domains:

	1	2
v_1	✓	✓
v_2	✓	
v_3	✓	✓

The REGULAR($\mathcal{M}, \langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle$) constraint specifies that for any solution A holds that $A(\langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle) \in L_{\mathcal{M}}$, that are, $A(\langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle) = 111$ and $A(\langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle) = 112$. Examples that violate the constraint are $A(\langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle) = 211$ and $A(\langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle) = 212$, as these inputs make the DFA terminate in the rejecting state q_{rej}.

As demonstrated by Katsirelos et al. (2009a), one important advantage of using an automaton based representation, over our encoding of GRAMMAR, is that it permits to compress the encoding using standard techniques for automaton minimisation (Hopcroft and Ullman, 1979).

Given a DFA $\mathcal{M}$ and a sequence of variables $\mathsf{v}_1, \ldots, \mathsf{v}_n$ representing an input string, we propose an encoding of REGULAR($\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$). Our encoding will represent all possible states the automaton can be in after processing the first i

symbols. Ultimately, an accepted input string must generate a sequence of transitions starting at the starting state and ending in some finite state.

Definition 5.11: ASP Encoding of REGULAR Constraints

For a DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ our ASP encoding of REGULAR($\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) is denoted by ASP-REGULAR[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$], and is constructed as follows:

- We introduce new atoms $\mathbf{q}_k(i)$ for each step i of $\mathcal{M}$'s processing, i.e., $0 \leq i \leq n$, and each state $q_k \in Q$, to indicate whether $\mathcal{M}$ is in state q_k after having processed the first i symbols from the input string.

- Each transition $\delta(q_j, t) = q_k$ is encoded by rules of the form

$$\mathbf{q}_k(i) \leftarrow \mathbf{q}_j(i-1),\ [[\mathsf{v}_i = t]] \tag{5.26}$$

  where $1 \leq i \leq n$. The intuitive meaning of a rule of the form (5.26) is that whenever $\mathcal{M}$ is in state q_j after having processed the first $i-1$ symbols and $\mathcal{M}$ reads t as the i-th symbol, then $\mathcal{M}$ transitions to the state q_k in step i.

- The condition that $\mathcal{M}$ must start processing in starting state q_0 is captured by the rule

$$\mathbf{q}_0(0) \leftarrow \tag{5.27}$$

  which sets $\mathbf{q}_0(0)$ unconditionally to true in any conflict-free assignment.

- To represent that the original constraint is violated if $\mathcal{M}$ finishes processing in a rejecting state $q_{rej} \in Q \setminus F$, we post rules of the form

$$[[\overline{\text{REGULAR}}(\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]] \leftarrow \mathbf{q}_{rej}(n) \tag{5.28}$$

  for each rejecting state q_{rej}.

The asymptotic space complexity of ASP-REGULAR[$\mathcal{G}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$] is $\mathcal{O}(|\mathcal{M}|)$, where $|\mathcal{M}|$ is the number of transitions.

Notice how the above encoding captures $\mathcal{M}$'s processing with rules of the single, straightforward and easily maintainable form (5.26). By allowing for a non-deterministic choice between the states in a transition, we could effortlessly go beyond DFA by representing non-deterministic finite automaton (NFA). NFAs can be significantly smaller than DFAs (Hopcroft and Ullman, 1979), but there is a price to

be paid: Because of their non-determinism, NFA-based models of REGULAR break with the one-to-one correspondence between the solutions for the encoding and the ones for the underlying constraint. Hence, we will the not further investigate their encoding in this thesis.

We proceed with demonstrating our DFA-based ASP encoding of REGULAR in the following example.

Example 5.7

Reconsider the DFA $\mathcal{M}$ from Example 5.6, with the input string of length 3, $\mathsf{v}_1\mathsf{v}_2\mathsf{v}_3$ with the domains $\mathrm{dom}(\mathsf{v}_1) = \mathrm{dom}(\mathsf{v}_3) = [1,2]$ and $\mathrm{dom}(\mathsf{v}_2) = [1,1]$. Our encoding ASP-REGULAR$[\mathcal{M}, \langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle]$ comprises the following rules:

$$
\begin{array}{ll}
\mathbf{q}_0(1) \leftarrow \mathbf{q}_0(0),\ [[\mathsf{v}_1 = 1]] & \mathbf{q}_{rej}(1) \leftarrow \mathbf{q}_1(0),\ [[\mathsf{v}_1 = 1]] \\
\mathbf{q}_0(2) \leftarrow \mathbf{q}_0(1),\ [[\mathsf{v}_2 = 1]] & \mathbf{q}_{rej}(2) \leftarrow \mathbf{q}_1(1),\ [[\mathsf{v}_2 = 1]] \\
\mathbf{q}_0(3) \leftarrow \mathbf{q}_0(2),\ [[\mathsf{v}_3 = 1]] & \mathbf{q}_{rej}(3) \leftarrow \mathbf{q}_1(2),\ [[\mathsf{v}_3 = 1]] \\
\mathbf{q}_1(1) \leftarrow \mathbf{q}_0(0),\ [[\mathsf{v}_1 = 2]] & \mathbf{q}_{rej}(1) \leftarrow \mathbf{q}_{rej}(0),\ [[\mathsf{v}_1 = 1]] \\
\mathbf{q}_1(2) \leftarrow \mathbf{q}_0(1),\ [[\mathsf{v}_2 = 2]] & \mathbf{q}_{rej}(2) \leftarrow \mathbf{q}_{rej}(1),\ [[\mathsf{v}_2 = 1]] \\
\mathbf{q}_1(3) \leftarrow \mathbf{q}_0(2),\ [[\mathsf{v}_3 = 2]] & \mathbf{q}_{rej}(3) \leftarrow \mathbf{q}_{rej}(2),\ [[\mathsf{v}_3 = 1]] \\
\mathbf{q}_1(1) \leftarrow \mathbf{q}_1(0),\ [[\mathsf{v}_1 = 2]] & \mathbf{q}_{rej}(1) \leftarrow \mathbf{q}_{rej}(0),\ [[\mathsf{v}_1 = 2]] \\
\mathbf{q}_1(2) \leftarrow \mathbf{q}_1(1),\ [[\mathsf{v}_2 = 2]] & \mathbf{q}_{rej}(2) \leftarrow \mathbf{q}_{rej}(1),\ [[\mathsf{v}_2 = 2]] \\
\mathbf{q}_1(3) \leftarrow \mathbf{q}_1(2),\ [[\mathsf{v}_3 = 2]] & \mathbf{q}_{rej}(3) \leftarrow \mathbf{q}_{rej}(2),\ [[\mathsf{v}_3 = 2]] \\
\mathbf{q}_0(0) \leftarrow & \\
[[\overline{\text{REGULAR}}(\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]] \leftarrow \mathbf{q}_{rej}(3) &
\end{array}
$$

Verify that, if we want to satisfy the REGULAR$(\mathcal{M}, \langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle)$ constraint, the application of UP discovers that $\mathcal{M}$ must not be in state q_1 at step 2 or state q_{rej} at step 3, but it does not prune any other value, i.e.,

$$
\left.\begin{array}{l}
\mathbf{T}[[\mathsf{v}_2 = 1]], \mathbf{F}\mathbf{q}_{rej}(3), \\
\mathbf{F}[[\mathsf{v}_2 = 2]], \mathbf{F}\mathbf{q}_1(2)
\end{array}\right\} \in \mathrm{UP}(\Delta_\Psi, \{\mathbf{F}[[\overline{\text{REGULAR}}(\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]]\})
$$

where $\Psi = \text{ASP-REGULAR}[\mathcal{M}, \langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle] \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3\}]$.

Observe that by encoding the processing of $\mathcal{M}$ into ASP, the state of the automaton underlying a REGULAR constraint, i.e., the state of the constraint propagator, is made transparent via (assignments to) atoms of the form $\mathbf{q}_j(i)$. Any ASP solver can use these in its advanced search heuristics. Moreover, the CONFLICTANALY-

SIS procedure in CDNL can exploit the processing of $\mathscr{M}$ and expose the implicit relationship between states. This can improve propagation of REGULAR.

Theorem 5.12

REGULAR($\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) is satisfiable if and only if there exists a solution $\mathbf{A}$ for $\Delta_{\text{ASP-REGULAR}[\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle] \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]}$ s.t. $[[\overline{\text{REGULAR}}(\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]] \in \mathbf{A}^{\mathbf{F}}$.

Proof. To begin, let $\Psi = \text{ASP-REGULAR}[\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle] \cup \text{ASP-VALUE}[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$. We show both implications of the proposition.

($\Rightarrow$) Suppose REGULAR($\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) is satisfiable, i.e., there exists an assignment to the variables $\omega = \mathsf{v}_1 \ldots \mathsf{v}_n$, say ω is assigned the string $t_1 \ldots t_n$, such that $\mathscr{M}$ transitions through a sequence of states $(q_{k_0}, \ldots, q_{k_n})$ starting at $q_{k_0} = q_0$ into an accepting state q_{k_n}. We can construct an assignment $\mathbf{A}$ the following way, for $1 \leq i \leq n$:

- $\mathbf{T}[[\mathsf{v}_i = t_i)]] \in \mathbf{A}$,
- $\mathbf{T}\mathbf{q}_{k_i}(i) \in \mathbf{A}$,
- all bodies are set to the value equivalent to the conjunction of their elements, and all other atoms are set to false.

We observe that no nogood in Δ_Ψ is contained in $\mathbf{A}$, i.e., $\mathbf{A}$ is conflict-free. Since $\mathbf{A}$ is also total, $\mathbf{A}$ is a solution for Δ_Ψ.

($\Leftarrow$) Let $\mathbf{A}$ be a solution for the set Δ_Ψ such that $[[\overline{\text{REGULAR}}(\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]] \in \mathbf{A}^{\mathbf{F}}$. We show that REGULAR($\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) is satisfiable. To begin, the nogood $\{\mathbf{F}\mathbf{q}_0(0)\}$, represented by rule (5.27), guarantees that $\mathbf{q}_0(0) \in \mathbf{A}^{\mathbf{T}}$.

Now, let q_j be any state such that $\mathbf{q}_j(i-1) \in \mathbf{A}^{\mathbf{T}}$, $1 \leq i \leq n$. Since $\mathbf{A}$ extends the assignment to the value encoding, we have $[[\mathsf{v}_i = t]] \in \mathbf{A}^{\mathbf{T}}$ for some value t, and the nogoods represented by rules of the form (5.26) force $\mathbf{q}_k(i) \in \mathbf{A}^{\mathbf{T}}$ for some state q_k with $\delta(q_j, t) = q_k$. We can continue this reasoning until reaching a final state q_ℓ, i.e., $\mathbf{q}_\ell(n) \in \mathbf{A}^{\mathbf{T}}$. The nogoods represented by rules r of the form (5.28), i.e., $\{\mathbf{F}[[\overline{\text{REGULAR}}(\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]], \mathbf{T}\,\text{body}(r)\}$ and $\{\mathbf{F}\,\text{body}(r), \mathbf{F}\mathbf{q}_\ell(n)\}$, guarantee the final state q_ℓ is accepting. Otherwise, $\mathbf{A}$ would be conflicting, contradicting the assumption.

In conclusion, all the visited atoms $\mathbf{q}_k(i)$ such that $\mathbf{q}_k(i) \in \mathbf{A}^{\mathbf{T}}$, starting from some $\mathbf{q}_0(0)$ and finishing ing $\mathbf{q}_\ell(n)$, form a processing of $\mathscr{M}$ whose transitions form a sequence satisfying REGULAR($\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$). □

To ensure that UP prunes all possible values, we extend our encoding by only a few more items. Following the general idea of Bacchus (2007), we introduce new atoms, each representing whether a particular transition is possible at a certain step in the DFA's processing. We will use these atoms to encode support for an input or a state leading to an accepting state.

Definition 5.12: Domain Consistent ASP Encoding of REGULAR Constraints

Our improved ASP encoding of the REGULAR($\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$) constraint is constructed as follows:

- Include ASP-REGULAR[$\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$].
- Support for the assignment $\mathsf{v}_i = t$ is given through a transition $\delta(q_j, t) = q_k$, i.e., from the state q_j to the state q_k at step i while reading t. In order to encode support, we define auxiliary atoms $\mathbf{d}(q_j, q_k, i)$ for each transition $\delta(q_j, t) = q_k$ by rules of the form

$$\mathbf{d}(q_j, q_k, i) \leftarrow \mathbf{q}_j(i-1),\ [[\mathsf{v}_i = t]],\ \mathbf{q}_k(i) \tag{5.29}$$

  where $1 \leq i \leq n$.

  Now, for an assignment $\mathsf{v}_i = t$, let (q_{j_ℓ}, q_{k_ℓ}) for $1 \leq \ell \leq m$ be the pairs of states such that $\delta(q_{j_i}, t) = q_{k_i}$, i.e., states connected when $\mathscr{M}$ reads t. We encode the existence of support for $\mathsf{v}_i = t$ by rules of the form

$$\leftarrow [[\mathsf{v}_i = t]],\ not\ \mathbf{d}(q_{j_1}, q_{k_1}, i),\ \ldots,\ not\ \mathbf{d}(q_{j_m}, q_{k_m}, i) \tag{5.30}$$

  where $1 \leq i \leq n$. A rule of the form (5.30) is satisfied if either the assignment $\mathsf{v}_i = t$ has a support, or the atom $[[\mathsf{v}_i = t]]$ is false.

- Finally, for each state q_j and step in $\mathscr{M}$'s processing, there must be an outgoing transition. Let $q_{k_1}, \ldots, q_{k_m}$ be transitions out of q_j. We encode the existence of a transition to a successor state for q_j by rules of the form

$$\leftarrow \mathbf{q}_j(i-1),\ not\ \mathbf{d}(q_j, q_{k_1}, i),\ \ldots,\ not\ \mathbf{d}(q_j, q_{k_m}, i) \tag{5.31}$$

  where i is a step in $\mathscr{M}$'s processing, $1 \leq i \leq n$.

We denote this improved encoding by ASP-REGULAR-DC[$\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$].

The space complexity of ASP-REGULAR-DC[$\mathscr{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$], in comparison to its

simpler counterpart, remains unchanged. It is still $\mathcal{O}(|\mathcal{M}|)$. Hence, there is no penalty in terms of asymptotic size of the encoding. Though, there are $\mathcal{O}(|Q|)$ rules of the form (5.31), where $|Q|$ is the number of states in $\mathcal{M}$, that have a rather large body, dependent on the number of incoming transitions.

Example 5.8

Reonsider the DFA $\mathcal{M}$ from Example 5.6, again applied to the string $\mathsf{v}_1\mathsf{v}_2\mathsf{v}_3$ with $\mathrm{dom}(\mathsf{v}_1) = \mathrm{dom}(\mathsf{v}_3) = [1,2]$ and $\mathrm{dom}(\mathsf{v}_2) = [1,1]$. In addition to the ones specified in Example 5.6, our ASP encoding ASP-REGULAR-DC$[\mathcal{M}, \langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle]$ comprises the following rules:

$$\mathbf{d}(q_0, q_0, 1) \leftarrow \mathbf{q}_0(0),\ [[\mathsf{v}_1 = 1]],\ \mathbf{q}_0(1)$$
$$\mathbf{d}(q_0, q_0, 2) \leftarrow \mathbf{q}_0(1),\ [[\mathsf{v}_2 = 1]],\ \mathbf{q}_0(2)$$
$$\mathbf{d}(q_0, q_0, 3) \leftarrow \mathbf{q}_0(2),\ [[\mathsf{v}_3 = 1]],\ \mathbf{q}_0(3)$$
$$\mathbf{d}(q_0, q_1, 1) \leftarrow \mathbf{q}_0(0),\ [[\mathsf{v}_1 = 2]],\ \mathbf{q}_1(1)$$
$$\mathbf{d}(q_0, q_1, 3) \leftarrow \mathbf{q}_0(2),\ [[\mathsf{v}_3 = 2]],\ \mathbf{q}_1(3)$$
$$\mathbf{d}(q_1, q_1, 1) \leftarrow \mathbf{q}_1(0),\ [[\mathsf{v}_1 = 2]],\ \mathbf{q}_1(1)$$
$$\mathbf{d}(q_1, q_1, 3) \leftarrow \mathbf{q}_1(2),\ [[\mathsf{v}_3 = 2]],\ \mathbf{q}_1(3)$$
$$\mathbf{d}(q_1, q_{rej}, 1) \leftarrow \mathbf{q}_1(0),\ [[\mathsf{v}_1 = 1]],\ \mathbf{q}_{rej}(1)$$
$$\mathbf{d}(q_1, q_{rej}, 2) \leftarrow \mathbf{q}_1(1),\ [[\mathsf{v}_2 = 1]],\ \mathbf{q}_{rej}(2)$$
$$\mathbf{d}(q_1, q_{rej}, 3) \leftarrow \mathbf{q}_1(2),\ [[\mathsf{v}_3 = 1]],\ \mathbf{q}_{rej}(3)$$
$$\mathbf{d}(q_{rej}, q_{rej}, 1) \leftarrow \mathbf{q}_{rej}(0),\ [[\mathsf{v}_1 = 1]],\ \mathbf{q}_{rej}(1)$$
$$\mathbf{d}(q_{rej}, q_{rej}, 2) \leftarrow \mathbf{q}_{rej}(1),\ [[\mathsf{v}_2 = 1]],\ \mathbf{q}_{rej}(2)$$
$$\mathbf{d}(q_{rej}, q_{rej}, 3) \leftarrow \mathbf{q}_{rej}(2),\ [[\mathsf{v}_3 = 1]],\ \mathbf{q}_{rej}(3)$$
$$\mathbf{d}(q_{rej}, q_{rej}, 1) \leftarrow \mathbf{q}_{rej}(0),\ [[\mathsf{v}_1 = 1]],\ \mathbf{q}_{rej}(1)$$
$$\mathbf{d}(q_{rej}, q_{rej}, 3) \leftarrow \mathbf{q}_{rej}(2),\ [[\mathsf{v}_3 = 2]],\ \mathbf{q}_{rej}(3)$$
$$\leftarrow [[\mathsf{v}_1 = 1]],\ \mathit{not}\ \mathbf{d}(q_0, q_0, 1),\ \mathit{not}\ \mathbf{d}(q_1, q_{rej}, 1),\ \mathit{not}\ \mathbf{d}(q_{rej}, q_{rej}, 1)$$
$$\leftarrow [[\mathsf{v}_2 = 1]],\ \mathit{not}\ \mathbf{d}(q_0, q_0, 2),\ \mathit{not}\ \mathbf{d}(q_1, q_{rej}, 2),\ \mathit{not}\ \mathbf{d}(q_{rej}, q_{rej}, 2)$$
$$\leftarrow [[\mathsf{v}_3 = 1]],\ \mathit{not}\ \mathbf{d}(q_0, q_0, 3),\ \mathit{not}\ \mathbf{d}(q_1, q_{rej}, 3),\ \mathit{not}\ \mathbf{d}(q_{rej}, q_{rej}, 3)$$
$$\leftarrow [[\mathsf{v}_1 = 2]],\ \mathit{not}\ \mathbf{d}(q_0, q_1, 1),\ \mathit{not}\ \mathbf{d}(q_1, q_1, 1),\ \mathit{not}\ \mathbf{d}(q_{rej}, q_{rej}, 1)$$
$$\leftarrow [[\mathsf{v}_3 = 2]],\ \mathit{not}\ \mathbf{d}(q_0, q_1, 3),\ \mathit{not}\ \mathbf{d}(q_1, q_1, 3),\ \mathit{not}\ \mathbf{d}(q_{rej}, q_{rej}, 3)$$

$$\leftarrow \mathbf{q}_0(0),\ not\ \mathbf{d}(q_0,q_0,1),\ not\ \mathbf{d}(q_0,q_1,1)$$
$$\leftarrow \mathbf{q}_0(1),\ not\ \mathbf{d}(q_0,q_0,2),\ not\ \mathbf{d}(q_0,q_1,2)$$
$$\leftarrow \mathbf{q}_0(2),\ not\ \mathbf{d}(q_0,q_0,3),\ not\ \mathbf{d}(q_0,q_1,3)$$
$$\leftarrow \mathbf{q}_1(0),\ not\ \mathbf{d}(q_1,q_1,1),\ not\ \mathbf{d}(q_1,q_{rej},1)$$
$$\leftarrow \mathbf{q}_1(1),\ not\ \mathbf{d}(q_1,q_1,2),\ not\ \mathbf{d}(q_1,q_{rej},2)$$
$$\leftarrow \mathbf{q}_1(2),\ not\ \mathbf{d}(q_1,q_1,3),\ not\ \mathbf{d}(q_1,q_{rej},3)$$
$$\leftarrow \mathbf{q}_{rej}(0),\ not\ \mathbf{d}(q_{rej},q_{rej},1)$$
$$\leftarrow \mathbf{q}_{rej}(1),\ not\ \mathbf{d}(q_{rej},q_{rej},2)$$
$$\leftarrow \mathbf{q}_{rej}(2),\ not\ \mathbf{d}(q_{rej},q_{rej},3)$$

Verify that, if we want to satisfy the REGULAR($\mathcal{M}$, $\langle \mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3 \rangle$) constraint, in addition to the effects of UP described in Example 5.7, UP immediately prunes the value 2 from the domain of v_1. Amongst other effects, UP discovers that only the transitions represented by the auxiliary atoms $\mathbf{d}(q_0,q_0,1)$, $\mathbf{d}(q_0,q_0,2)$, $\mathbf{d}(q_0,q_0,3)$, and $\mathbf{d}(q_0,q_1,3)$ may satisfy the constraint. The other auxiliary atoms are assigned **F**.

Note that rules of the form (5.30) are not neccessary to achieve correctness, i.e., rules of the form (5.29–5.31) are redundant. However, they ensure that UP can prune all possible values. In fact, encoding the REGULAR constraint into ASP by means of a DFA yields a result similar to Theorem 5.8, our approach that is based on production rules.

Theorem 5.13: Domain Consistency on REGULAR via Unit Propagation

UP on ASP-REGULAR-DC[$\mathcal{M}$, $\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$] $\cup$ ASP-VALUE[$\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$] enforces domain consistency on REGULAR($\mathcal{M}$, $\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$).

Proof. The proof strategy is to show that if an assignment to a variable is possible then there is a combination of values to the other variables that satisfies the constraint. To improve readability, let Ψ = ASP-REGULAR-DC[$\mathcal{M}$, $\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$] $\cup$ ASP-VALUE[$\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$].

Let **A** be an assignment to the atoms in a value representation of the variables in the scope of the constraint, i.e., the atoms in the program ASP-VALUE[$\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}$], representing the current set of values in the domain of the variables. Suppose UP has been run to completion, extending $\mathbf{A} \cup \{\mathbf{F}[[\overline{\text{REGULAR}}(\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]]\}$ (i.e., the REGULAR constraint shall not be violated) to the assignment $\mathbf{A}'$.

Consider any variable v_i in the scope of the constraint and a value t that has not been removed from the domain of v_i. That means $[[\mathsf{v}_i = t]] \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., $[[\mathsf{v}_i = t]]$ is either true or unassigned. Then, the nogood $\{\mathbf{T}[[\mathsf{v}_i = t]]\} \cup \{\mathbf{Fd}(q_j, q_k, i) \mid \delta(q_j, t) = q_k \in \mathscr{M}\}$ represented by a rule of the form (5.30) guarantees that a transition $\delta(q_j, t) = q_k$ is possible, i.e., $\mathbf{d}(q_j, q_k, i) \notin (\mathbf{A}')^{\mathbf{F}}$. In turn, there is a rule r of the form (5.29) representing nogoods $\{\mathbf{Td}(q_j, q_k, i), \mathbf{F}\,\mathrm{body}(r)\} \in \mathbf{\Delta}_{\Psi}^{\mathbf{d}(q_j,q_k,i)} \subseteq \Delta_{\Psi}$, and $\{\mathbf{T}\,\mathrm{body}(r), \mathbf{Fq}_j(i-1)\}, \{\mathbf{T}\,\mathrm{body}(r), \mathbf{Fq}_k(i)\} \in \mathrm{EQ}_{\mathrm{body}(r)} \subseteq \Delta_{\Psi}$ that guarantee $\mathbf{q}_j(i-1), \mathbf{q}_k(i) \notin (\mathbf{A}')^{\mathbf{F}}$. That is, reaching state q_j at step $i-1$ and q_k at step i is allowed.

It remains to show that $\mathscr{M}$ can in fact reach state q_j at step $i-1$ from the starting state, and an accepting state from state q_k at step i.

First, let q_j be any state of $\mathscr{M}$ such that $\mathbf{q}_j(i) \notin (\mathbf{A}')^{\mathbf{F}}$ where $1 \leq i \leq n$. The transition function defines the set of predecessor states $q_{j_1}, \ldots, q_{j_m}$ for q_j, that are, states q_{j_ℓ} such that $\delta(q_{j_\ell}, t) = q_j$ for some value t where $1 \leq \ell \leq m$. In our encoding, we have the rules $\mathrm{ES}_{\Psi}(\{\mathbf{q}_k(i)\}) = \{r_1, \ldots, r_m\}$, i.e., rules with head $\mathbf{q}_k(i)$, given through

$$
\begin{aligned}
r_1: &\quad \mathbf{q}_j(i) \leftarrow \mathbf{q}_{j_1}(i-1),\ [[\mathsf{v}_i = t_{j_1}]] \\
&\ \ldots \\
r_m: &\quad \mathbf{q}_j(i) \leftarrow \mathbf{q}_{j_m}(i-1),\ [[\mathsf{v}_i = t_{j_m}]]
\end{aligned}
$$

for each step i. The nogood $\{\mathbf{F}\,\mathrm{body}(r_1), \ldots, \mathbf{F}\,\mathrm{body}(r_m), \mathbf{Tq}_k(i)\} \in \mathbf{\Delta}_{\Psi}^{\mathbf{q}_j(i)} \subseteq \Delta_{\Psi}$ guarantees that at least one transition $\delta(q_{j_\ell}, t_{j_\ell}) = q_j$ is applicable, i.e., $\mathrm{body}(r_\ell) \notin (\mathbf{A}')^{\mathbf{F}}$. Then, the nogoods $\{\mathbf{T}\,\mathrm{body}(r_\ell), \mathbf{F}[[\mathsf{v}_i = t_{j_\ell}]]\}$ and $\{\mathbf{T}\,\mathrm{body}(r_\ell), \mathbf{Fq}_{j_\ell}(i-1)\}$ from $\mathrm{EQ}_{\mathrm{body}(r_\ell)}$ guarantee $[[\mathsf{v}_i = t_{j_\ell}]], \mathbf{q}_{j_\ell}(i-1) \notin (\mathbf{A}')^{\mathbf{F}}$. In other words, $\mathscr{M}$ can reach the state q_{j_ℓ} at step $i-1$ and read the symbol t_{j_ℓ}, i.e., the assignment $\mathsf{v}_i = t_{j_\ell}$ is possible. We continue this reasoning until reaching the starting state. Since $\mathbf{q}_0(0) \in (\mathbf{A}')^{\mathbf{F}}$ by the nogood $\{\mathbf{Fq}_0(0)\} \in \mathbf{\Delta}_{\Psi}^{\mathbf{q}_0(0)} \subseteq \Delta_{\Psi}$ represented by rule (5.27), we have a possible processing of $\mathscr{M}$ reaching state q_j at step $i-1$.

Second, let q_k be any state of $\mathscr{M}$ such that $\mathbf{q}_k(i) \notin (\mathbf{A}')^{\mathbf{F}}$ where $1 \leq k < n$. Then, the nogood $\{\mathbf{Tq}_k(i)\} \cup \{\mathbf{Fd}(q_k, q_\ell, i+1) \mid \delta(q_k, t) = q_\ell\}$ represented by a rule of the form (5.31) guarantees that a transition $\delta(q_k, t) = q_\ell$ is possible, i.e., $\mathbf{d}(q_k, q_\ell, i+1) \notin (\mathbf{A}')^{\mathbf{F}}$. In turn, there is a rule r of the form (5.29) representing the nogoods

$$
\begin{aligned}
&\{\mathbf{Td}(q_k, q_\ell, i+1), \mathbf{F}\,\mathrm{body}(r)\} \in \mathbf{\Delta}_{\Psi}^{\mathbf{d}(q_k,q_\ell,i+1)}, \\
&\{\mathbf{T}\,\mathrm{body}(r), \mathbf{F}[[\mathsf{v}_{i+1} = t']]\}, \text{ and } \{\mathbf{T}\,\mathrm{body}(r), \mathbf{Fq}_\ell(i+1)\} \in \mathrm{EQ}_{\mathrm{body}(r)}
\end{aligned}
$$

for some value t', that guarantee $[[\mathsf{v}_{i+1} = t']], \mathbf{q}_\ell(i+1) \notin (\mathbf{A}')^{\mathbf{F}}$. That is, the value t' has not been removed from the domain of v_{i+1} and reaching the state q_ℓ at step

$i+1$ is allowed. We can continue this reasoning until reaching a final state. Then, since $[[\overline{\text{REGULAR}}(\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]] \in (\mathbf{A}')^{\mathbf{F}}$ the nogoods represented by rules r of the form (5.28), i.e., $\{\mathbf{F}[[\overline{\text{REGULAR}}(\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle)]], \mathbf{T}\,\text{body}(r)\}$ and $\{\mathbf{F}\,\text{body}(r), \mathbf{Fq}_\ell(n)\}$, guarantee that a final, accepting state q_ℓ can be reached.

In conclusion, whenever $\mathsf{v}_i = t$ one can find compatible values in the domains of all the other variables. Since at least one value must be in each domain, guaranteed by (4.2), we have a set of non-empty domains which are domain consistent. □

With our improved encoding, UP provides an efficient domain consistency propagator for free. Note that the specialised algorithm of Sellmann (2006) and the SAT-based method of Bacchus (2007) have a similar asymptotic run time complexity.

Theorem 5.14

UP on ASP-REGULAR-DC$[\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle] \cup$ ASP-VALUE$[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$ runs in $\mathcal{O}(nd|\mathcal{M}|)$ time down any branch of the search tree

Proof. For each of the n variables, there are $\mathcal{O}(d)$ constant-size nogoods and one nogood of size $\mathcal{O}(d)$ represented by ASP-VALUE$[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$. On the other hand, ASP-REGULAR-DC$[\mathcal{M}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle]$ represents $\mathcal{O}(n|\mathcal{M}|)$ nogoods of constant size.

By Theorems 2.1 we obtain, down any branch of the search tree, a total running time complexity given by $\mathcal{O}(nd) + \mathcal{O}(n|M|) \leq \mathcal{O}(nd|M|)$. □

Experimental Results

We tested the practical utility of our ASP encodings on a set of shift-scheduling instances. The benchmark stems from Côté et al. (2011) and has been previously used by Quimper and Walsh (2007) and Katsirelos et al. (2009b) to compare the performance of CYK-based GRAMMAR constraint propagator in CP and SAT systems. The problem is to schedule employees in a company to activities subject to the following rules.

- An employee either works on activity a_i, has a break b, has lunch l, or rests r.
- If the company business is open, an employee works on an activity for a minimum of one hour and can change activities after a fifteen minutes break or one hour lunch.
- Break and lunch both are scheduled between periods of work.

– A part-time employee works at least three hours and at most six hours plus a fifteen minutes break, while

– a full-time employee works at least six hours and at most eight hours plus an hour and a half for the lunch and the breaks.

Our goal is to minimise the number of hours worked.

The schedule of an employee is modelled with a sequence of 96 variables, each encodes a time slot of 15 minutes and can take a value that represents whether the employee works on an activity, has a break, has lunch, or rests. The sequence must be produced by the following CFG, encoding above rules, with restrictions on productions.

$$
\begin{aligned}
S &::= RFR,\ \mathsf{f}_F(i,j) \equiv 30 \leq j \leq 38 \\
S &::= RPR,\ \mathsf{f}_F(i,j) \equiv 13 \leq j \leq 24 \\
F &::= PLP \\
P &::= WbW \\
W &::= A_i,\ \mathsf{f}_W(i,j) \equiv j \geq 4 \\
L &::= lL \mid l,\ \mathsf{f}_L(i,j) \equiv j = 4 \\
A_i &::= a_i A_i \mid a_i,\ \mathsf{f}_{A_i}(i,j) \equiv open(i) \\
R &::= rR \mid r
\end{aligned}
$$

In addition to the GRAMMAR constraint on each of the m employee's schedule, we also post a constraint $\sum_m [[v_i^m = a_i]] \geq \mathbf{d}(t, a_i)$ in order to satisfy the demand $\mathbf{d}(t, a_i)$ for each activity a_i at time t. To break symmetry, we force the schedules for the employees to be in lexicographical order.

A *bottom-up* ASP grounder such as *gringo* (3.0.3)[7] can be employed to generate our encodings. (*Bottom-up* describes the fashion of the grounding process, i.e., instantiating a program from a first-order signature by systematically substituting all occurrences of first-order variables with terms, starting from facts and continuing with rules following positive predicate dependency.; cf. Gebser et al., 2011b) Then, the grounder simulates a CYK parser, i.e., it constructs all possible parsings for all possible subsequences of the input sequence. However, similar to the CYK parser, it also generates productions that cannot produce the starting symbol S. This represents a significant overhead. Instead, we have implemented a grounder

[7] `http://potassco.sourceforge.net/`

Table 5.5: Results on shift scheduling instances.

$\|A\|$	#	m	ASP-GRAMMAR	ASP-GRAMMAR-DC	SAT
1	2	4	**26.00**	26.25	**26.00**
1	3	6	**37.25**	37.50	37.50
1	4	6	**38.00**	**38.00**	**38.00**
1	5	5	**24.00**	**24.00**	**24.00**
1	6	6	**33.00**	**33.00**	**33.00**
1	7	8	**49.00**	**49.00**	**49.00**
1	8	3	**20.50**	**20.50**	**20.50**
1	10	9	**54.00**	54.25	54.25
2	1	5	**25.00**	**25.00**	**25.00**
2	2	10	**58.00**	58.75	59.25
2	3	6	**39.50**	40.25	**39.50**
2	9	3	**19.00**	**19.00**	**19.00**
2	4	11	**68.25**	68.50	69.00
2	5	4	**24.50**	24.75	25.50
2	6	5	**28.25**	29.25	28.50
2	8	5	**32.00**	32.75	32.25
2	10	8	57.25	57.75	**57.00**

for the special purpose of this benchmark based on the algorithm in (Quimper and Walsh, 2007).

We consider our two ASP encodings of the GRAMMAR constraint: ASP-GRAMMAR and ASP-GRAMMAR-DC. The latter allows UP to achieve domain consistency. We compare against the SAT model by Quimper and Walsh (2007). Experiments were run with the ASP solver *clasp* (1.3.5)[7] on a 2.00 GHz PC under Linux, where each run was limited to 3600 s time and 1 GByte RAM.

Table 5.2 presents our results on instances involving one or two activities. $|A|$ denotes the number of activities, # denotes the problem number, and m denotes the number of employees. For each setting, the table provides the best solution in the number of worked hours (lower is better) that was found by the ASP solver within 3600 s time.

Regardless of the encoding, the solver returned a feasible solution for all instances after a few seconds, and rarely reported an improved result within the remaining run time. Optimality was only reported on four instances. No clear conclusion can be drawn comparing the three models, i.e., no encoding performs significantly better that the other. We therefore advocate our ASP-GRAMMAR encoding because it is simpler, more straightforward, easier to maintain, and com-

petes with the other approaches on the benchmark domain. As the SAT encoding is reported to outperform a CP model of the problem (Quimper and Walsh, 2007), we expect that our ASP encoding beat the CP approach just as well.

5.3 The REACHABILITY Constraint

Since our ASP encodings of constraints presented in the previous sections are all tight, it was reasonable to limit our studies on the impact of ASP inference to UP. We here turn our attention to conditions of REACHABILITY (cf. Dooms et al., 2005) that are straightforwardly encoded into non-tight programs. We make several contributions to this line of research.

- First, we show how to represent the graph variables' and vertex set variables' domains with ASP. Then, we proceed with studying the impact of UP and unfounded set inference on propagating REACHABILITY with varying degrees on freedom.

- Because ASP allows for recursive definitions and employs very efficient inference mechanisms such as UP and WFN, it should naturally and efficiently handle REACHABILITY. Whilst this intuition is strengthened by experimental results (Celik et al., 2009; Coban et al., 2008), we demonstrate that restricting inference to the combination of UP and WFN can hinder propagation in general.

- Additional information, however, can be drawn from unfounded sets. We show that BL and LD can lead to additional pruning, and that the complete range of ASP inference considered in this thesis maintains domain consistency of REACHABILITY.

- We demonstrate that under certain limitations, however, a restricted set of inference operators is sufficient to achieve a similar result. In particular, we prove that if the graph and source nodes are fixed, then UP and FL prune all values. If the graph and source nodes are not fixed, but the set of reached nodes are, then UP and BL prune all values.

- We provide theoretical results on asymptotic run time complexity.

- Finally, we experimentally evaluate the effects of ASP inference on benchmarks that make use of REACHABILITY conditions.

The (global) REACHABILITY constraint works on graph- and vertex set variables. As outlined in the Background chapter, we will make some practical assumptions on the structure of their domains, following Dooms et al. (2005).

Assumption: Graph Variables' and Vertex Set Variables' Domains

The domain of a graph variable G is the lattice of graphs $[lb(\mathsf{G}), ub(\mathsf{G})]$. The domain of a vertex set variable X is $[lb(X), ub(X)]$, the powerset of the set of vertices $ub(X)$ in a graph under inclusion of the set of vertices $lb(X) \subseteq ub(X)$.

Given this, we define the REACHABILITY constraint as follows.

Definition 5.13: REACHABILITY

For an assignment A, the REACHABILITY constraint REACHABILITY(G, S, T) is satisfied if and only if G is a graph such that T is the set of nodes reachable from some node in S.

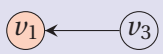
Reachability is a relevant condition in many applications, in particular, applications that require a graph to be connected. For instance, in the problem of placing valves in an urban hydraulic network, we want to maximise the sub-net that *stays connected to a water source* when a pipe is isolated for maintenance (see Section 6.4). Another example is the Hidato puzzle presented earlier.

Example 5.9

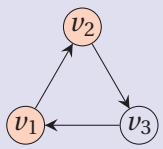
Consider the REACHABILITY(G, S, T) constraint where the domain of G is given through

$$lb(\mathsf{G}) = (\{v_1, v_3\}, \{(v_3, v_1)\}) \text{ and}$$
$$ub(\mathsf{G}) = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}),$$

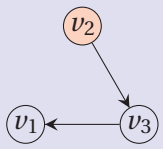
the domain of S is $[\emptyset, \{v_1, v_2\}]$, and the domain of T is $[\{v_1\}, \{v_1, v_2, v_3\}]$. The graph $lb(\mathsf{G})$ looks like this:



and the graph $ub(\mathsf{G})$ looks like this:



The potential start vertices $v_1, v_2 \in ub(S)$ are lightly shaded red. As can be seen from the domain of G, the vertices v_1 and v_3, and the edge (v_3, v_1) is included in every possible assignment to G. An assignment that satisfies the constraint is A with $A(\mathsf{G}) = (\{v_1, v_3\}, \{(v_3, v_1)\})$, $A(S) = \{v_1\}$, and $A(T) = \{v_1\}$. Another satisfying assignment is A' with $A'(\mathsf{G}) = (\{v_1, v_2, v_3\}, \{(v_2, v_3), (v_3, v_1)\})$, $A'(S) = \{v_2\}$, and $A'(T) = \{v_1, v_2, v_3\}$. The graph $A'(\mathsf{G})$ looks like this:



Contrary to the intuition that ASP systems naturally and efficiently handle REACHABILITY, we demonstrate that they in fact hinder its propagation, as ASP systems restrict inference to a combination of UP and WFN. Additional information, however, can be drawn from unfounded sets. We show that LD can lead to additional pruning, and that applying UP, FL, BL, and LD on REACHABILITY prunes all possible values.

To begin, we show how to represent the domain of a graph variable G and vertex set variables S and T. For encoding G we introduce a new atom for each vertex $v \in ub(\mathsf{G})$ and for each edge $(u, v) \in ub(\mathsf{G})$. We will denote these atoms $[\![v \in \mathsf{G}]\!]$ and $[\![(u, v) \in \mathsf{G}]\!]$, respectively. An assignment to $[\![v \in \mathsf{G}]\!]$ will indicate whether the vertex v belongs to G. Hence, $[\![v \in \mathsf{G}]\!]$ will be assigned true if $v \in \mathsf{G}$, and false if the vertex v has been removed from the domain of G. Similarly, an assignment to $[\![(u, v) \in \mathsf{G}]\!]$ will indicate whether the edge (u, v) belongs to G, and $[\![(u, v) \in \mathsf{G}]\!]$ will be assigned true if the edge (u, v) belongs to G, and false if it has been removed from the domain of G. Extra care has to be taken when encoding a consistent set of domains. In particular, an edge (u, v) can only be included in G if both vertices u and v belong to G.

Definition 5.14: ASP Encoding of Graph Variables' Domains

Our ASP encoding of a graph variable G's domain, denoted by ASP-GRAPH[G], is specified as follows.

– For each pair of distinct vertices $u, v \in ub(\mathsf{G})$, include rules of the form

$$\{\ [\![v \in \mathsf{G}]\!]\ \} \leftarrow \tag{5.32}$$

$$\{ [\![(u,v) \in \mathsf{G}]\!] \} \leftarrow [\![v \in \mathsf{G}]\!],\ [\![v \in \mathsf{G}]\!] \tag{5.33}$$

– For each vertex $v \in lb(\mathsf{G})$, include the integrity constraint

$$\leftarrow not\ [\![v \in \mathsf{G}]\!] \tag{5.34}$$

– Similarly, for each edge $(u,v) \in lb(\mathsf{G})$, include the integrity constraint

$$\leftarrow not\ [\![(u,v) \in \mathsf{G}]\!] \tag{5.35}$$

The choice rules from (5.32–5.33) represent the upper bound on the domain of a graph variable G, where a rule of the form (5.32) encodes the possible inclusion of a vertex, and a rule of the form (5.33) encodes the possible inclusion of an edge (u,v) if both vertices u and v are included. On the other hand, the remaining integrity constraints (5.34–5.35) represent the lower bound on the domain of G, where a rule of the form (5.34) enforces the inclusion of a vertex, and a rule of the form (5.35) enforces the inclusion of an edge.

Example 5.10

Reconsider the setting from Example 5.9 where the domain of G was given through

$$lb(\mathsf{G}) = (\{v_1, v_3\}, \{(v_3, v_1)\}) \text{ and}$$
$$ub(\mathsf{G}) = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}).$$

Our encoding of G, ASP-GRAPH[G], is

$$\{ [\![v_1 \in \mathsf{G}]\!] \} \leftarrow \qquad \{ [\![(v_1, v_2) \in \mathsf{G}]\!] \} \leftarrow [\![v_1 \in \mathsf{G}]\!],\ [\![v_2 \in \mathsf{G}]\!]$$
$$\{ [\![v_2 \in \mathsf{G}]\!] \} \leftarrow \qquad \{ [\![(v_2, v_3) \in \mathsf{G}]\!] \} \leftarrow [\![v_2 \in \mathsf{G}]\!],\ [\![v_3 \in \mathsf{G}]\!]$$
$$\{ [\![v_3 \in \mathsf{G}]\!] \} \leftarrow \qquad \{ [\![(v_3, v_1) \in \mathsf{G}]\!] \} \leftarrow [\![v_3 \in \mathsf{G}]\!],\ [\![v_1 \in \mathsf{G}]\!]$$

$$\leftarrow not\ [\![v_1 \in \mathsf{G}]\!]$$
$$\leftarrow not\ [\![v_3 \in \mathsf{G}]\!]$$
$$\leftarrow not\ [\![(v_3, v_1) \in \mathsf{G}]\!]$$

which can be compiled into a program without choice rules. From this, we can construct the set of nogoods $\Delta_{\text{ASP-GRAPH}[\mathsf{G}]}$. Verify that UP on ASP-GRAPH[G]

achieves an assignment **A** that reflects G's domain. In particular, for the vertices we have $\mathbf{T}[[v_1 \in \mathsf{G}]], \mathbf{T}[[v_3 \in \mathsf{G}]] \in \mathbf{A}$ and $[[v_2 \in \mathsf{G}]] \notin \mathbf{A^T} \cup \mathbf{A^F}$. For the edges, we have $\mathbf{T}[[(v_3, v_1) \in \mathsf{G}]] \in \mathbf{A}$ and $[[(v_1, v_2) \in \mathsf{G}]], [[(v_2, v_3) \in \mathsf{G}]] \notin \mathbf{A^T} \cup \mathbf{A^F}$.

To expand on the example, the removal of the vertex v_2 from G is indicated by $\mathbf{F}[[v_2 \in \mathsf{G}]] \in \mathbf{A}$. In turn, the removal of the edges $(v_1, v_2), (v_2, v_3)$ from G is represented by $\mathbf{F}[[(v_1, v_2) \in \mathsf{G}]], \mathbf{F}[[(v_2, v_3) \in \mathsf{G}]] \in \mathbf{A}$.

$lb(\mathsf{G})$	$ub(\mathsf{G})$	Property of **A**
$(\{v_1\},\emptyset)$	$(\{v_1, v_3\},\{(v_3, v_1)\})$	$\mathbf{T}[[v_1 \in \mathsf{G}]] \in \mathbf{A}, [[v_3 \in \mathsf{G}]], [[(v_3, v_1) \in \mathsf{G}]] \notin \mathbf{A^T} \cup \mathbf{A^F}$
$(\{v_1\},\emptyset)$	$(\{v_1\},\emptyset)$	$\mathbf{T}[[v_1 \in \mathsf{G}]], \mathbf{F}[[v_2 \in \mathsf{G}]], \mathbf{F}[[v_3 \in \mathsf{G}]] \in \mathbf{A}$

The table relates sample domains of G with properties that can be observed in an assignment **A** reflecting G.

Encoding the vertex set variables S and T is straightforward, using the atoms from ASP-GRAPH[G]. In general, for a vertex set variable X, we introduce a new atom for each vertex $v \in ub(X)$. We will denote this atom $[[v \in X]]$. An assignment to $[[v \in X]]$ will indicate the inclusion or exclusion of v. Hence, $[[v \in X]]$ will be assigned true if v is included in X, and false if v is excluded from X.

Definition 5.15: ASP Encoding of Vertex Set Variables' Domains

Our encoding of a vertex set variable X' domain, denoted by ASP-VSET[G, X], is specified as follows.

– For each vertex $v \in ub(X)$, include a rule of the form

$$\{\ [[v \in X]]\ \} \leftarrow [[v \in \mathsf{G}]] \tag{5.36}$$

– For each vertex $v \in lb(X)$, include the integrity constraint

$$\leftarrow not\ [[v \in X]] \tag{5.37}$$

Obviously, any non-empty program ASP-VSET[G, X] has externals in ASP-GRAPH[G], that are, atoms representing the inclusion of vertices into G. The choice rules from (5.36) represent the upper bound on the domain of a vertex set variable X, where each rule of the form (5.36) encodes the possible inclusion of a vertex if it is a vertex in the underlying graph. On the other hand, the integrity constraints from (5.37) represent the lower bound on the domain of X, where each rule of the form (5.37) enforces the inclusion of a vertex.

We use ASP-VSET[G, S] to represent the domain of S and ASP-VSET[G, T] to represent the domain of T.

Example 5.11

Reconsider the setting from Example 5.9 where the domain of S is $[\emptyset, \{v_1, v_2\}]$, and the domain of T is $[\{v_1\}, \{v_1, v_2, v_3\}]$. Our encoding of the vertex sets S and T, i.e., ASP-VSET[G, S] and ASP-VSET[G, T], is

$$\{ [\![v_1 \in S]\!] \} \leftarrow [\![v_1 \in \mathsf{G}]\!] \qquad \{ [\![v_1 \in T]\!] \} \leftarrow [\![v_1 \in \mathsf{G}]\!]$$
$$\{ [\![v_2 \in S]\!] \} \leftarrow [\![v_2 \in \mathsf{G}]\!] \qquad \{ [\![v_2 \in T]\!] \} \leftarrow [\![v_2 \in \mathsf{G}]\!]$$
$$\leftarrow not\ [\![v_1 \in T]\!] \qquad \{ [\![v_3 \in T]\!] \} \leftarrow [\![v_3 \in \mathsf{G}]\!]$$

which can be compiled into a program without choice rules. Given this, we can construct the set of nogoods $\Delta_{\text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, X]}$.

Verify that UP achieves an assignment $\mathbf{A}$ that reflects S' and T's domains. In particular, we have $\mathbf{T}[\![v_1 \in T]\!] \in \mathbf{A}$ and $[\![v_i \in T]\!], [\![v_{i-1} \in S]\!] \notin \mathbf{A^T} \cup \mathbf{A^F}$ for $2 \leq i \leq 3$. Some other examples are given in the table below.

v_1	v_2	v_3	Property of $\mathbf{A}$
✓	✗		$\mathbf{T}[\![v_1 \in \mathsf{G}]\!], \mathbf{F}[\![v_2 \in \mathsf{G}]\!] \in \mathbf{A}, [\![v_3 \in \mathsf{G}]\!] \notin \mathbf{A^T} \cup \mathbf{A^F}$
✓	✗	✓	$\mathbf{T}[\![v_1 \in \mathsf{G}]\!], \mathbf{F}[\![v_2 \in \mathsf{G}]\!], \mathbf{T}[\![v_3 \in \mathsf{G}]\!] \in \mathbf{A}$

The table relates membership information of the vertices v_1, v_2, and v_3 to T (✓ inclusion, ✗ exclusion) with properties that can be observed in an assignment $\mathbf{A}$ reflecting T.

Note that our encoding of graph variables' and vertex set variables' domains is rather technical. In practice, the domains of some of the variables may be fixed, allowing for simpler encodings. For instance, if the graph G is fixed then the edges in G are typically encoded by using facts rather than choice rules.

Now that we have settled on a representation of graph structures, we can proceed with encoding REACHABILITY conditions. The following encoding of REACHABILITY is folklore (cf. Niemelä, 1999; Lifschitz, 2008a; Brewka et al., 2011).

Definition 5.16: ASP Encoding of REACHABILITY Constraints

Our ASP encoding of the REACHABILITY(G, S, T) constraint is constructed as follows:

– For each vertex $v \in ub(\mathsf{G})$ we introduce the new atom **reached**(v) to

reflect whether a vertex v is reached.

– Then, include the following set of rules, where $v, w \in ub(\mathsf{G})$ are distinct vertices.

$$\textbf{reached}(v) \leftarrow [[v \in S]] \tag{5.38}$$

$$\textbf{reached}(v) \leftarrow \textbf{reached}(u),\ [[(u,v) \in \mathsf{G}]] \tag{5.39}$$

– Finally, the constraint is violated if the reached vertices do not coincide with the ones specified in the vertex set T, captured by the rules of the form

$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v \in T]],\ not\ \textbf{reached}(v) \tag{5.40}$$

$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \textbf{reached}(v),\ not\ [[v \in T]] \tag{5.41}$$

where $v \in ub(\mathsf{G})$.

We denote the encoding by ASP-REACHABILITY$[\mathsf{G}, S, T]$.

The encoding is intuitive: Rules of the from (5.38) and (5.39) encode that a vertex v is reached if it is a start vertex or, alternatively, if there is an edge to v from another vertex u where u is reached. The remaining rules of the form (5.40) and (5.41) ensure that REACHABILITY(G, S, T) is violated if the value of T does not correspond to the set of vertices that are reached.

Note that our encoding allows for some flexibility. For instance, in practice one might only want to enforce that T represents a subset of reachable nodes rather than a one-to-one correspondence. Then, all rules of the form (5.41) may be dropped. In total, ASP-REACHABILITY$[\mathsf{G}, S, T] \cup$ ASP-GRAPH$[\mathsf{G}] \cup$ ASP-VSET$[\mathsf{G}, S] \cup$ ASP-VSET$[\mathsf{G}, T]$ introduces $\mathcal{O}(n^2)$ new atoms and $\mathcal{O}(n^2)$ rules, where n is the number of vertices in $ub(\mathsf{G})$.

We proceed with investigating the impact of ASP inference on our encoding in terms of local consistency achieved on the original constraint. We will start with the special case where G and S are fixed, i.e., G and S work as parameters. Then, the constraint ASP-REACHABILITY$[\mathsf{G}, S, T]$ amounts to queries about vertices that can be reached from S in G.

Example 5.12

Consider the REACHABILITY(G, S, T) constraint, that we do not want to vio-

late, and the assignment **A** representing the fixed domains of the graph variable $\mathsf{G} = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{(v_1, v_2), (v_3, v_4), (v_4, v_3)\}$), and the start vertices $S = \{v_1\}$, and the domain of T given through $lb(T) = \emptyset$ and $ub(T) = \{v_1, v_2, v_3, v_4\}$. In particular, we have

$$\left.\begin{array}{l}\mathbf{F}[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]],\ \mathbf{T}[[v_1 \in S]],\\ \mathbf{T}[[(v_1, v_2) \in \mathsf{G}]],\ \mathbf{T}[[(v_3, v_4) \in \mathsf{G}]],\ \mathbf{T}[[(v_4, v_3) \in \mathsf{G}]]\end{array}\right\} \in \mathbf{A}.$$

The value of G looks like this:

The single start vertex v_1 is lightly shaded red. Observe that the vertices v_3 and v_4 are disconnected from v_1. The program ASP-REACHABILITY$[\mathsf{G}, S, T]$ includes the following rules:

$$\mathbf{reached}(v_1) \leftarrow [[v_1 \in S]]$$

$$\mathbf{reached}(v_2) \leftarrow \mathbf{reached}(v_1),\ [[(v_1, v_2) \in \mathsf{G}]]$$
$$\mathbf{reached}(v_3) \leftarrow \mathbf{reached}(v_4),\ [[(v_4, v_3) \in \mathsf{G}]]$$
$$\mathbf{reached}(v_4) \leftarrow \mathbf{reached}(v_3),\ [[(v_3, v_4) \in \mathsf{G}]]$$

$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_1 \in T]],\ \mathit{not}\ \mathbf{reached}(v_1)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_2 \in T]],\ \mathit{not}\ \mathbf{reached}(v_2)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_3 \in T]],\ \mathit{not}\ \mathbf{reached}(v_3)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_4 \in T]],\ \mathit{not}\ \mathbf{reached}(v_4)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \mathbf{reached}(v_1),\ \mathit{not}\ [[v_1 \in T]]$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \mathbf{reached}(v_2),\ \mathit{not}\ [[v_2 \in T]]$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \mathbf{reached}(v_3),\ \mathit{not}\ [[v_3 \in T]]$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \mathbf{reached}(v_4),\ \mathit{not}\ [[v_4 \in T]]$$

We omit the remaining rules that do not apply, given the above domains. Considering the assignment **A**, immediate consequences of UP are $v_1, v_2 \in lb(T)$, i.e.,

$$\text{UP}^*(\Delta_\Psi, \mathbf{A}) = \mathbf{A}' \supseteq \{\mathbf{T}[[v_1 \in T]], \mathbf{T}[[v_2 \in T]]\}$$

where

$$\Psi = \text{ASP-REACHABILITY}[\mathsf{G}, S, T] \cup \text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T].$$

On the other hand, FL determines that v_3 and v_4 are not reachable, i.e.,

$$\text{WFN}^*[\text{loop}(\Psi)](\Psi, \mathbf{A}') = \mathbf{A}'' \supseteq \{\mathbf{F}\ \textbf{reached}(v_3), \mathbf{F}\ \textbf{reached}(v_4)\}.$$

Further application of UP removes v_3 and v_4 from the domain of T, i.e.,

$$\text{UP}^*(\Delta_\Psi, \mathbf{A}'') \supseteq \{\mathbf{F}[[v_3 \in T]], \mathbf{F}[[v_4 \in T]]\}.$$

In conclusion, we have $T = \{v_1, v_2\}$.

In general, if G and S are fixed, then the application of UP and FL inference on $\text{ASP-REACHABILITY}[\mathsf{G}, S, T]$ prunes all values of T.

Theorem 5.15

If G and S are fixed, then UP and FL on

$$\Psi = \text{ASP-REACHABILITY}[\mathsf{G}, S, T] \cup \text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T].$$

achieve domain consistency on $\text{REACHABILITY}(\mathsf{G}, S, T)$.

Proof. The proof strategy is to show that if an assignment to a variable is possible then there is a combination of values to the other variables that satisfies the constraint. Since G and S are fixed, we will prove for values of T, i.e., the exclusion of vertices from T and the inclusion of vertices to T.

Let $\mathbf{A}$ be an assignment to the atoms representing the current domains of G, S, and T, i.e., atoms from $\text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T]$. Since G and S are fixed, $\mathbf{A}$ is total for $\text{atom}(\text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S])$. Suppose UP and FL have been run to completion, extending $\mathbf{A} \cup \mathbf{F}[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]]$ (the REACHABILITY constraint shall not be violated) to the conflict-free assignment $\mathbf{A}'$.

Consider any vertex $v \in ub(T)$. We make a case distinction between $[[v \in T]] \notin (\mathbf{A}')^{\mathbf{T}}$ and $[[v \in T]] \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., the exclusion of v from T or the inclusion of v to T is possible.

First, if $[[v \in T]] \notin (\mathbf{A}')^{\mathbf{T}}$, then the nogoods $\{\mathbf{F}[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]], \mathbf{T}\,\text{body}(r)\}$ and $\{\mathbf{F}\,\text{body}(r), \mathbf{T}\ \textbf{reached}(v), \mathbf{F}[[v \in T]]\}$ represented by a rule r of the form (5.41)

guarantee that $\mathbf{reached}(v) \notin (\mathbf{A}')^{\mathbf{T}}$. In turn, the nogoods $\{\mathbf{F}\,\mathbf{reached}(v), \mathbf{T}\,\mathrm{body}(r)\}$ and $\{\mathbf{F}\,\mathrm{body}(r), \mathbf{T}[[v \in S]]\}$ represented by a rule r of the form (5.38) guarantee that $[[v \in S]] \notin (\mathbf{A}')^{\mathbf{T}}$. (Since $\mathbf{A}'$ is total for atom(ASP-VSET$[\mathsf{G}, S]$) and conflict-free, we have $[[v \in S]] \in (\mathbf{A}')^{\mathbf{F}}$, i.e., $v \notin S$.) Similarly, for every incident edge $(u, v) \in \mathsf{G}$ we have $[[(u, v) \in \mathsf{G}]] \in (\mathbf{A}')^{\mathbf{T}}$ and a rule r of the form (5.39) representing nogoods $\{\mathbf{T}\,\mathrm{body}(r), \mathbf{F}\,\mathbf{reached}(v)\}$ and $\{\mathbf{F}\,\mathrm{body}(r), \mathbf{T}\,\mathbf{reached}(u), \mathbf{T}[[(u, v) \in \mathsf{G}]]\}$. This guarantees that we have $\mathbf{reached}(u) \notin (\mathbf{A}')^{\mathbf{T}}$. (Since $\mathbf{A}'$ is total for atom(ASP-VSET$[\mathsf{G}, S]$) and conflict-free, we have $[[u \in S]] \in (\mathbf{A}')^{\mathbf{F}}$, i.e., $u \notin S$.) That means, every predecessor u of v can be disconnected. By successively applying the same argument, we obtain paths (including loops), all of which contain only vertices that can be disconnected. For each such vertex u, the nogoods $\{\mathbf{F}[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]], \mathbf{T}\,\mathrm{body}(r)\}$ and $\{\mathbf{F}\,\mathrm{body}(r), \mathbf{T}\,\mathbf{reached}(u), \mathbf{F}[[u \in T]]\}$ represented by a rule r of the form (5.41) guarantee that $[[u \in T]] \notin (\mathbf{A}')^{\mathbf{T}}$, i.e., the exclusion of v from T is possible. In conclusion, if $[[v \in T]] \notin (\mathbf{A}')^{\mathbf{T}}$, i.e., the exclusion of v from T is possible, then there are compatible exclusions of other vertices that satisfy the constraint. (In fact, since G is fixed, there is no degree of freedom, and the other vertices are indeed excluded from T.)

On the other hand, if $[[v \in T]] \notin (\mathbf{A}')^{\mathbf{F}}$, then the nogoods represented by a rule r of the form (5.40), i.e., $\{\mathbf{F}[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]], \mathbf{T}\,\mathrm{body}(r)\}$ and $\{\mathbf{F}\,\mathrm{body}(r), \mathbf{T}[[v \in T]], \mathbf{F}\,\mathbf{reached}(v)\}$, guarantee that $\mathbf{reached}(v) \notin (\mathbf{A}')^{\mathbf{F}}$. Since FL ran to completion, $\mathbf{A}'$ is unfounded-free. Hence, $\mathrm{ES}_{\text{ASP-REACHABILITY}[\mathsf{G},S,T]}(\{\mathbf{reached}(v)\}) \setminus \mathbf{A}^{\mathbf{F}} \neq \emptyset$. That means, either $\mathbf{F}\{[[v \in S]]\} \notin \mathbf{A}'$ or $\mathbf{F}\{\mathbf{reached}(u), [[(u, v) \in \mathsf{G}]]\} \notin \mathbf{A}'$ for some $(u, v) \in \mathsf{G}$, i.e., either $v \in S$ or v has a predecessor u that is reached. By successively applying the same argument, we obtain a path which concludes in a start vertex. Hence, if $[[v \in T]] \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., the inclusion of v into T is possible, then there are compatible inclusions of other vertices that satisfy the constraint. (In fact, since G is fixed, there is no degree of freedom, and the other vertices are indeed included in T.)

In conclusion, if the exclusion of v from T or the inclusion of v to T is possible, then there is a combination of values to the other variables that satisfies the constraint. Hence, we have a set of domains which are domain consistent. □

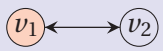
We now turn our attention to another special case of REACHABILITY(G, S, T), i.e., the set of reachable vertices in T is fixed. This is useful in situations where access of a vertex to a resource, represented through start vertices, has to be guaranteed. In such circumstances, applying UP and WFN on ASP-REACHABILITY$[\mathsf{G}, S, T]$ can hinder propagation, in general. The construction of a counter example is simple.

Counter Example 5.13

Consider the REACHABILITY(G, S, T) constraint, that we do not want to violate, and the assignment $\mathbf{A}$ representing the fixed domain of $T = \{v_1, v_2\}$, and the domains of the graph variable G given through

$$lb(\mathsf{G}) = (\{v_1, v_2\}, \emptyset) \text{ and}$$
$$ub(\mathsf{G}) = (\{v_1, v_2\}, \{(v_1, v_2), (v_2, v_1)\}),$$

and the start vertices S given through $lb(S) = \emptyset$ and $ub(S) = \{v_1\}$. In particular, we have $\mathbf{reached}(v_1), \mathbf{reached}(v_2) \in \mathbf{A^T}$. The graph $ub(\mathsf{G})$ looks like this:



The single (potential) start vertex $v_1 \in ub(S)$ is lightly shaded red. Our encoding ASP-REACHABILITY$[\mathsf{G}, S, T]$ includes the following rules:

$$\mathbf{reached}(v_1) \leftarrow [\![v_1 \in S]\!]$$
$$\mathbf{reached}(v_2) \leftarrow [\![v_2 \in S]\!]$$
$$\mathbf{reached}(v_1) \leftarrow \mathbf{reached}(v_2),\ [\![(v_2, v_1) \in \mathsf{G}]\!]$$
$$\mathbf{reached}(v_2) \leftarrow \mathbf{reached}(v_1),\ [\![(v_1, v_2) \in \mathsf{G}]\!]$$

$$[\![\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]\!] \leftarrow [\![v_1 \in T]\!],\ not\ \mathbf{reached}(v_1)$$
$$[\![\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]\!] \leftarrow [\![v_2 \in T]\!],\ not\ \mathbf{reached}(v_2)$$
$$[\![\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]\!] \leftarrow \mathbf{reached}(v_1),\ not\ [\![v_1 \in T]\!]$$
$$[\![\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]\!] \leftarrow \mathbf{reached}(v_2),\ not\ [\![v_2 \in T]\!]$$

We get $(v_2, v_1) \in lb(\mathsf{G})$ as an immediate consequence of UP, i.e.,

$$\text{UP}^*(\Delta_\Psi, \mathbf{A}) \ni \mathbf{T}[\![(v_2, v_1) \in \mathsf{G}]\!]$$

where

$$\Psi = \text{ASP-REACHABILITY}[\mathsf{G}, S, T] \cup \text{ASP-GRAPH}[\mathsf{G}]$$
$$\cup\ \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T].$$

However, verify that neither UP nor WFN on Ψ include v_1 in the lower bound of S, represented by $\mathbf{T}[\![v_1 \in S]\!]$.

However, we can guarantee that the addition of WFJ inference prunes all values.

Theorem 5.16

If T is fixed then UP and BL on

$$\Psi = \text{ASP-REACHABILITY}[\mathsf{G}, S, T] \cup \text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T].$$

achieve domain consistency on $\text{REACHABILITY}(\mathsf{G}, S, T)$.

Proof. The proof strategy is to show that if an assignment to a variable is possible then there is a combination of values to the other variables that satisfies the constraint. Since T is fixed, we will prove for values of G and S, i.e., the exclusion of edges from G and the inclusion of edges to G, and the exclusion of vertices from S and the inclusion of vertices to S.

Let $\mathbf{A}$ be an assignment to the atoms representing the current domains of G, S, and T, i.e., atoms from $\text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T]$. Since T is fixed, $\mathbf{A}$ is total for $\text{atom}(\text{ASP-VSET}[\mathsf{G}, T])$. Suppose UP and BL have been run to completion, extending $\mathbf{A} \cup \mathbf{F}[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]]$ (i.e., the REACHABILITY constraint shall not be violated) to the conflict-free assignment $\mathbf{A}'$.

Consider any edge $(u, v) \in ub(\mathsf{G})$. We make a case distinction between $[[(u, v) \in \mathsf{G}]] \notin (\mathbf{A}')^{\mathbf{T}}$ and $[[(u, v) \in \mathsf{G}]] \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., the exclusion of (u, v) from G or the inclusion of (u, v) to G is possible. First, let $[[(u, v) \in \mathsf{G}]] \notin (\mathbf{A}')^{\mathbf{F}}$. If $\mathbf{reached}(u) \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., $u \in T$ is possible, then the nogoods $\{\mathbf{F}\,\text{body}(r), \mathbf{T}\ \mathbf{reached}(u), \mathbf{T}[[(u, v) \in \mathsf{G}]]\}$ and $\{\mathbf{F}\ \mathbf{reached}(v), \mathbf{T}\,\text{body}(r)\}$ represented by a rule r of the form (5.39) guarantee that $\mathbf{reached}(v) \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., $v \in T$ is possible. (In fact, since T is fixed, there is no degree of freedom, and if $u \in T$ then $v \in T$.) Hence, the inclusion of the edge (u, v) into G does not connect a vertex that is reached with a disconnected one. In conclusion, there is an assignment with $(u, v) \in \mathsf{G}$ satisfying the constraint.

On the other hand, let $[[(u, v) \in \mathsf{G}]] \notin (\mathbf{A}')^{\mathbf{T}}$. If $\mathbf{reached}(v) \in (\mathbf{A}')^{\mathbf{T}}$, i.e., $v \in T$ via a rule of the form (5.41), then

$$\text{ES}_{\text{ASP-REACHABILITY}[\mathsf{G},S,T]}(\{\mathbf{reached}(v)\}) \setminus \mathbf{A}^{\mathbf{F}} \neq \{\{\mathbf{reached}(u), [[(u, v) \in \mathsf{G}]]\}\}$$

i.e., if v is reached then either $v \in S$ is possible or there is some other edge (u', v) that can connect a reachable vertex u' to v. (In fact, since T is fixed, there is no degree of freedom, and if $v \in T$ then $u' \in T$.) By successively applying the same ar-

gument, we obtain paths, each of which concludes in a vertex that can be included in S. Hence, there is an assignment with $(u, v) \notin \mathsf{G}$ satisfying the constraint.

In conclusion, if the exclusion of (u, v) from G or the inclusion of (u, v) to G is possible, then there is a combination of values to the other variables that satisfies the constraint.

Now, consider any vertex $v \in ub(S)$. We make a case distinction between $[\![v \in S]\!] \notin (\mathbf{A}')^{\mathbf{T}}$ and $[\![v \in S]\!] \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., the exclusion of v from S or the inclusion of v to S is possible. First, if $[\![v \in S]\!] \notin (\mathbf{A}')^{\mathbf{F}}$ then the nogoods $\{\mathbf{F}\,\mathrm{body}(r), \mathbf{T}[\![v \in S]\!]\}$ and $\{\mathbf{F}\,\mathbf{reached}(v), \mathbf{T}\,\mathrm{body}(r)\}$ represented by a rule r of the form (5.38) guarantee that $\mathbf{reached}(v) \notin (\mathbf{A}')^{\mathbf{F}}$, i.e., $v \in T$ is possible via a rule of the form (5.41). (In fact, since T is fixed, there is no degree of freedom, and $v \in T$.) Hence, the inclusion of the vertex v into S does not connect a vertex that is disconnected. Hence, the constraint is satisfied.

On the other hand, let $[\![v \in S]\!] \notin (\mathbf{A}')^{\mathbf{T}}$. If $\mathbf{reached}(v) \in (\mathbf{A}')^{\mathbf{T}}$, i.e., $v \in T$ via a rule of the form (5.41), then

$$\mathrm{ES}_{\text{ASP-REACHABILITY}[\mathsf{G},S,T]}(\{\mathbf{reached}(v)\}) \setminus \mathbf{A}^{\mathbf{F}} \neq \{\{[\![v \in S]\!]\}\}$$

i.e., if v is reached then v being a start vertex is not the only way to connect v, i.e., there is some edge (u, v) that can connect a reachable vertex u to v where

$$\mathrm{ES}_{\text{ASP-REACHABILITY}[\mathsf{G},S,T]}(\{\mathbf{reached}(v)\}) \setminus \mathbf{A}^{\mathbf{F}} \ni \{\mathbf{reached}(u), [\![(u, v) \in \mathsf{G}]\!]\}.$$

From $\{\mathbf{reached}(u), [\![(u, v) \in \mathsf{G}]\!]\} \notin (\mathbf{A}')^{\mathbf{F}}$ and the nogoods in $\mathrm{EQ}_{\{\mathbf{reached}(u),[\![(u,v)\in\mathsf{G}]\!]\}}$ we conclude $\mathbf{reached}(u), [\![(u, v) \in \mathsf{G}]\!] \notin (\mathbf{A}')^{\mathbf{F}}$. (In fact, we have $\mathbf{reached}(u) \in (\mathbf{A}')^{\mathbf{T}}$ via a rule of the form (5.41) since T is fixed, and there is no degree of freedom.) Hence, if v is reached then there is an edge that can be included in G connecting a reachable vertex to v. By successively applying the same argument, we obtain paths, each of which concludes in a vertex that can be included in S, satisfying the constraint.

In conclusion, if the exclusion of v from S or the inclusion of v to S is possible, then there is a combination of values to the other variables that satisfies the constraint. Hence, we have a set of domains which are domain consistent. □

Example 5.14

Reconsider the setting from Counter Example 5.13.

Since $\{\mathbf{reached}(v_1), \mathbf{reached}(v_2)\} \in \text{loop}(\text{ASP-REACHABILITY}[\mathsf{G}, S, T])$ and

$$\text{ES}_{\text{ASP-REACHABILITY}[\mathsf{G},S,T]}(\{\mathbf{reached}(v_1), \mathbf{reached}(v_2)\}) \setminus \mathbf{A}^{\mathbf{F}} = \{\{[\![v_1 \in S]\!]\}\},$$

the application of BL enforces $\mathbf{T}\{[\![v_1 \in S]\!]\}$. In turn, UP includes v_1 in the lower bound of S, represented by $\mathbf{T}[\![v_1 \in S]\!]$.

If the value of T is not fixed, however, domain consistency is not guaranteed. Again, the construction of a counter example is simple.

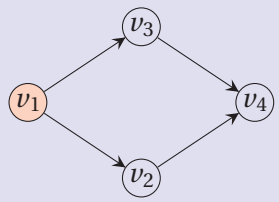
Counter Example 5.15

Consider the REACHABILITY(G, S, T) constraint, that we do not want to violate, and the assignment $\mathbf{A}$ representing the domains of the graph variable G given through

$$lb(\mathsf{G}) = (\{v_1, v_2, v_3, v_4\}, \emptyset) \text{ and}$$
$$ub(\mathsf{G}) = (\{v_1, v_2, v_3, v_4\}, \{(v_1, v_2), (v_1, v_3), (v_2, v_4), (v_3, v_4)\}),$$

the set variable of start vertices S given through $lb(S) = \emptyset$ and $ub(S) = \{v_1\}$, and the set variable of reachable vertices T given through $lb(T) = \{v_4\}$ and $ub(T) = \{v_1, v_2, v_3, v_4\}$. In particular, we have $\mathbf{T}\ \mathbf{reached}(v_4) \in \mathbf{A}$.

The graph $ub(\mathsf{G})$ looks like this:



The single (potential) start vertex $v_1 \in ub(S)$ is lightly shaded red. Our encoding ASP-REACHABILITY$[\mathsf{G}, S, T]$ includes the following rules:

$$\mathbf{reached}(v_1) \leftarrow [\![v_1 \in S]\!]$$
$$\mathbf{reached}(v_2) \leftarrow \mathbf{reached}(v_1),\ [\![(v_1, v_2) \in \mathsf{G}]\!]$$
$$\mathbf{reached}(v_3) \leftarrow \mathbf{reached}(v_1),\ [\![(v_1, v_3) \in \mathsf{G}]\!]$$
$$\mathbf{reached}(v_4) \leftarrow \mathbf{reached}(v_2),\ [\![(v_2, v_4) \in \mathsf{G}]\!]$$
$$\mathbf{reached}(v_4) \leftarrow \mathbf{reached}(v_3),\ [\![(v_3, v_4) \in \mathsf{G}]\!]$$

$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_1 \in T]],\ not\ \textbf{reached}(v_1)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_2 \in T]],\ not\ \textbf{reached}(v_2)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_3 \in T]],\ not\ \textbf{reached}(v_3)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow [[v_4 \in T]],\ not\ \textbf{reached}(v_4)$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \textbf{reached}(v_1),\ not\ [[v_1 \in T]]$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \textbf{reached}(v_2),\ not\ [[v_2 \in T]]$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \textbf{reached}(v_3),\ not\ [[v_3 \in T]]$$
$$[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]] \leftarrow \textbf{reached}(v_4),\ not\ [[v_4 \in T]]$$

We here omit the remaining rules that do not apply, given the above domains. Verify that neither UP nor WFN, nor WFJ include v_1 in the lower bound of S or T, represented by $\mathbf{T}[[v_1 \in S]]$ and $\mathbf{T}\ \textbf{reached}(v_1)$, respectively.

Additional pruning is required. We can show that if UP, FL, BL, and LD are available, then we can prune all values.

Theorem 5.17

UP, FL, BL and LD on

$$\Psi = \text{ASP-REACHABILITY}[\mathsf{G}, S, T] \cup \text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T].$$

achieve domain consistency on $\text{REACHABILITY}(\mathsf{G}, S, T)$.

Proof. The proof strategy is to show that if an assignment to a variable is possible then there is a combination of values to the other variables that satisfies the constraint. We need to consider this for values of G, S, and T, i.e., the exclusion of edges from G and the inclusion of edges to G, the exclusion of vertices from S and the inclusion of vertices to S, and the exclusion of vertices from T and the inclusion of vertices to T. We will prove the exclusion of vertices to T in detail. The other possible assignments follow the arguments from the proof of Theorems 5.15 and 5.16.

Let $\mathbf{A}$ be an assignment to the atoms representing the current domains of G, S, and T, i.e., atoms from $\text{ASP-GRAPH}[\mathsf{G}] \cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T]$. Suppose UP and BL have been run to completion, extending $\mathbf{A} \cup \mathbf{F}[[\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]]$ (i.e., the REACHABILITY constraint shall not be violated) to the conflict-free assignment $\mathbf{A}'$.

For any edge $(u, v) \in ub(\mathsf{G})$, the proof follows the one for Theorem 5.16, i.e., UP ensures that if (u, v) can be included in G then (u, v) does not connect a vertex that is reached via one that cannot be reached, and BL and UP guarantee that if (u, v) can be excluded from G then, if v can be included in T, there is either some path connecting a vertex to v that can be included in S, or v can be included in S itself. Hence, in any case, the constraint is satisfied.

Similarly, following the proof of Theorem 5.16, for any vertex $v \in ub(S)$, UP ensures that if v can be included in S then v can also be included in T, and BL and UP guarantee that if v can be excluded from S then, if v can be included in T, there is some path connecting a vertex to v that can be included in S. Hence, in any case, the constraint is satisfied.

Moreover, following the proof of Theorem 5.15, for any vertex $v \in ub(T)$, UP and FL ensure that if v can be included in T then either v can also be included in S, or there is some path connecting a vertex to v that can be included in S. On the other hand, UP guarantees that if v can be excluded from T then v can be excluded from S and every path (including loops) that ends in v contains only vertices that can be made disconnected from start vertices. In order to satisfy the constraint, however, it remains to show that, then, each other vertex w (distinct from v) that is included in T does not get disconnected, i.e., not all paths from a potential start vertex go though v. Let $w \in ub(T)$ such that w is included in T, i.e., $[\![v \in T]\!] \in (\mathbf{A}')^{\mathbf{T}}$. Then the nogoods $\{\mathbf{F}[\![\overline{\text{REACHABILITY}}(\mathsf{G}, S, T)]\!], \mathbf{T}\,\text{body}(r)\}$ and $\{\mathbf{F}\,\text{body}(r), \mathbf{T}[\![v \in T]\!], \mathbf{F}\,\mathbf{reached}(v)\}$ represented by a rule r of the form (5.40) guarantee that $\mathbf{reached}(v) \in (\mathbf{A}')^{\mathbf{T}}$. By construction of ASP-REACHABILITY$[\mathsf{G}, S, T]$, every set of vertices $V \subseteq ub(\mathsf{G})$, in particular every such set V with $w \in V$, induces a loop $\{\mathbf{reached}(v) \mid v \in V\}$ in the program ASP-REACHABILITY$[\mathsf{G}, S, T]$. Since LD ran to completion, we have

$$\begin{aligned}&\text{ES}_{\text{ASP-REACHABILITY}[\mathsf{G},S,T]}(\{\mathbf{reached}(v) \mid v \in V\}) \setminus \mathbf{A}^{\mathbf{F}}\\ &\qquad \not\subseteq \{\text{body}(r) \mid r \in \text{ASP-REACHABILITY}[\mathsf{G}, S, T],\ [\![v \in T]\!] \in \text{body}(r)\}\end{aligned}$$

for each set of vertices V such that $\{v\} \subseteq V \subseteq ub(\mathsf{G})$. Hence, not all paths from a potential start vertex go though v, and the exclusion of v from T does not disconnect vertices that are already included in T. Hence, the constraint is satisfied.

In conclusion, if the exclusion of v from T or the inclusion of v to T is possible, then there is a combination of values to the other variables that satisfies the constraint. Hence, we have a set of domains which are domain consistent. □

Example 5.16

Reconsider the setting from Example 5.15. Since the singleton $\{\mathbf{reached}(v_1)\}$ is a loop in loop(ASP-REACHABILITY[G, S, T]) and

$$\begin{aligned}&\mathrm{ES}_{\text{ASP-REACHABILITY}[\mathsf{G},S,T]}(\{\mathbf{reached}(v_1)\}) \setminus \mathbf{A}^{\mathbf{F}} = \{\mathrm{body}(r_2), \mathrm{body}(r_3)\}\\ &\subseteq \{\mathrm{body}(r) \mid r \in \text{ASP-REACHABILITY}[\mathsf{G}, S, T],\ \mathbf{reached}(v_1) \in \mathrm{body}(r)\},\end{aligned}$$

LD includes $\mathbf{reached}(v_1)$ in the lower bound of T, represented by the literal $\mathbf{T}\ \mathbf{reached}(v_1)$. In turn, the application of BL enforces $\mathbf{T}\{[\![v_1 \in S]\!]\}$, and UP includes v_1 in the lower bound of S, represented by $\mathbf{T}[\![v_1 \in S]\!]$.

Finally, we address the run time for propagating REACHABILITY conditions.

Theorem 5.18

UP, FL, BL and LD on

$$\begin{aligned}\Psi = &\text{ASP-REACHABILITY}[\mathsf{G}, S, T] \cup \text{ASP-GRAPH}[\mathsf{G}]\\ &\cup \text{ASP-VSET}[\mathsf{G}, S] \cup \text{ASP-VSET}[\mathsf{G}, T].\end{aligned}$$

run in $\mathcal{O}(n^2)$ time down any branch of the search tree.

Proof. There are $\mathcal{O}(n^2)$ nogoods represented by Ψ. By Theorems 2.1 we obtain, down any branch of the search tree, a running time complexity of $\mathcal{O}(n^2)$ for UP. Since ASP-REACHABILITY[G, S, T] is component-unary, by Theorem 3.7 and Theorem 3.12 a $\mathcal{O}(n)$ algorithm to compute dominators in the support-flowgraph representation of our encoding can simulate the application of BL and LD. Hence, the overall time complexity to enforce domain consistency on REACHABILITY(G, S, T) is given by $\mathcal{O}(n^2) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$. □

Experimental Results

An experimental evaluation of the effects of ASP inference on REACHABILITY conditions relies on the availability of BL and LD inference in ASP systems. To date, however, there is no ASP solver that implements a method for BL and LD. We have put forward a method in Chapter 3 that can be used to simulate the effects of BL and LD on REACHABILITY via computing dominators in the support-flowgraph of our encoding. Implementing Georgiadis and Tarjan (2004)'s linear-time algorithm for finding all dominators in a flowgraph, however, is a challenging engineering exercise as it relies on sophisticated data structures. Hence, for practical reasons, we

have integrated BL into the ASP solver *clasp* (2.1.1)[8] via failed-literal-detection and FL. This has high computational costs, i.e., down any branch of the search tree, cubic in the size of the program, whilst our flowgraph-based method has worst-case quadratic costs.

To compare with the state-of-the-art, i.e., limiting ASP inference to only use UP and FL, we include the default setting of *clasp* in our analysis. We conducted experiments on search problems that make use of REACHABILITY conditions. Our benchmarks stem from the Second ASP Competition (Denecker et al., 2009). Most of the problems also formed a benchmark class in the third ASP competition (Calimeri et al., 2011).

Connected Dominating Set

A *dominating set* in an undirected graph $\mathsf{G} = (V, E)$ is a subset $D \subseteq V$ of vertices in the graph such that for every vertex $v \in V$, either $v \in D$, or v is adjacent to a vertex $u \in D$, i.e., there is an edge $(u, v) \in E$. The dominating set D is *connected* if the sub-graph induced by the vertices in D is connected. Connected dominating sets are useful for efficient routing in ad hoc wireless networks (Wu and Li, 1999). Given a graph G and an integer k, the *connected dominating set* problem is to find a connected dominating set with at most k vertices.

Generalised Slitherlink

Slitherlink is logic puzzle game that is played on a grid of dots forming squares inside them. Some of the squares contain a number. The goal of the puzzle is to connect adjacent dots horizontally and vertically such that the lines form a loop, where the number in a square constraints how many of its four sides represent a segment in the loop. The *generalised Slitherlink* problem extends the grid to a graph problem, constraining the membership of subsets of edges in the loop.

Graph Partitioning

A *partitioning* of an undirected graph (V, E) is a labelling of the vertices V with the integers in $[1, k]$ such that the sub-graph induced by the vertices sharing a label, called a *partition*, is connected. Given the number of partitions k, a weight-function $w : E \rightarrow \mathbb{N}$ that assigns a weight to each edge in E, and a maximum weight w_{max}, the *graph partitioning* problem is to find a partitioning such that

[8] `http://potassco.sourceforge.net/`

Table 5.6: Experimental Data.

Benchmark				*UP+FL*				*UP+FL+BL*	
Class	n	s	time	branches	conflicts	s	time	branches	conflicts
Connected Dominating Set	21	**20**	**202**	11321k	6339k	**20**	3342	**6887k**	**3655k**
Generalised Slitherlink	29	**29**	**3**	22k	5k	**29**	4	**1k**	**<1k**
Graph Partitioning	13	**13**	**147**	3159k	2345k	**13**	785	**1139k**	**810k**
Hamiltonian Path	29	**29**	**1**	44k	18k	**29**	8	**6k**	**3k**
Maze Generation	29	**26**	**53**	3832k	1906k	20	1700	**1426k**	**881k**

for every pair of partitions the sum of the weights of the edges connecting a vertex from each partition is at-most w_{max}.

Hamiltonian Graph

A *Hamiltonian path* is a path in a directed graph $\mathsf{G} = (V, E)$ that passes every vertex $v \in V$ exactly once. The Hamiltonian graph problem is to find a Hamiltonian path in a given graph.

Maze Generation

A maze is a tour puzzle played on a two-dimensional grid. Each cell in the grid either contains a wall or is empty. The goal is to find a path from a distinguished entry cell to a distinguished exit cell. The *maze generation* problem is to create a maze subject to criteria that make the maze a realistic one. For instance, every empty cell in the maze must be reachable from the entry.

Summary of Results

The following definitions apply to Table 5.6 of results. *UP+FL* denotes *clasp*'s default setting, and *UP+FL+BL* denotes the setting that integrates BL. In each benchmark class, n denotes the total number of instances and s denotes the number of instances for which the program terminated within the allowed time. Most importantly, the table shows the time taken to compute all instances in the class that

were solved in both settings. Similarly, we provide the total number of branches and the number of conflicts during search, aggregated over all instances in the class that were solved in both settings. Experiments were run on a Linux PC, where each run was limited to 1200s time on a single 2.00 GHz core and 1 GByte RAM.

From the results shown in Table 5.6, it can be concluded that information from BL prunes search dramatically: The additional propagation in *UP+FL+BL* decreases the number of branches and conflicts by roughly one order of magnitude in comparison to *UP+FL*. On the other hand, the high computational costs of propagating BL via failed-literal-detection are clearly reflected in the run times of *UP+FL+BL*. These costs, however, could be drastically reduced by using Georgiadis and Tarjan (2004)'s linear-time algorithm, and by making the computation of dominators incremental. In conclusion, our experiments encourage the implementation of these techniques.

5.4 Limitations

In the previous sections, we have presented ASP encodings for some important constraints and shown that the inference of an ASP solver on our encodings of GRAMMAR, REGULAR, and REACHABILITY constraints can achieve domain consistency with an asymptotic run time complexity that is similar to specialised algorithms propagating the same constraints. It might be supposed that, there are many more constraint propagators that can be simulated by ASP inference on a simple, straightforward, and easily maintainable logic programming model. In fact, Gent's *Fundamental Conjecture of Reformulation* suggests that it may always be possible to transform a constraint into a set of smaller constraints for which propagation can be performed establishing the same level of consistency in the same asymptotic run time as a specialised propagator for the original constraint (Gent, 2001).

Gent's conjecture, however, was refuted by Bessière et al. (2009a) showing that it is impossible to encode ALL-DIFFERENT into SAT using polynomial space such that UP achieves domain consistency. Though, a domain consistency constraint propagator exists (Régin, 1994). As ASP solving shares many techniques with SAT, including UP inference, it is reasonable to assume that Bessière et al.'s result also carries over to ASP. For certain, there is no practical, tight ASP encoding of ALL-DIFFERENT that will enable UP inference to prune all values in general, given the close relationship between tight programs and SAT. Though, it remains an open

question whether the effect of unfounded set inference on a non-tight encoding can help to simulate a domain consistent propagator of ALL-DIFFERENT, but the author of this thesis is doubtful.

Whilst it may not always be possible to translate constraint propagation into ASP inference on a logic programming encoding that achieves a strong form of local consistency, our experimental analysis has shown little evidence to support a claim that using those strong translations are beneficial. Moreover, we have found them harder to encode and not easy to maintain. On the other hand, our experimental results have shown that there is hardly any penalty involved in using the simpler, easily maintainable encodings instead.

In fact, the greater advantages of our translation-based approach to constraint answer set solving over competing ones are provided through the conflict-driven techniques it enables. In our approach, for instance, CONFLICTANALYSIS can exploit constraint interdependencies which, in turn, can improve propagation between constraints and contribute to the dynamic search heuristic modern ASP solvers employ. Albeit dynamic search heuristics are intriguing, our approach lacks the possibility to include specialised heuristics, or even a search algorithms, into CASP specifications as it is common in CP systems. Problem-specific search heuristic often lead to high computational impact. Recent developments in incorporating problem-specific search heuristics into ASP (Gebser et al., 2013b), however, may be extended to CASP.

A key limitation our translation-based approach to constraint answer set solving is the asymptotic space complexity of our ASP encodings. When constraints range over variables with large domains, our encoding become large, challenging computational resources even if the asymptotic space complexity is a small polynomial. Another limitation, that also plagues SAT, is that the best case asymptotic run time (down any branch of the search tree) of UP on our ASP encodings is the same as the worst case.

5.5 Related Work

There is some related work that empirically compares of the performance of ASP and CP systems on solving CSP. An extensive study conducted by Dovier et al. (2005) on concludes that ASP encodings are more compact, and more declarative than CP encodings. There was no clear winner in terms of computational efficiency, as the performance of the respective solvers was affected by the na-

ture of the specific problem. ASP systems were particularly scalable on graph-based problems, but performance deteriorated on problems with arithmetic constraints or large domains, creating an issue with encoding size. Coban et al. (2008) conducted an empirical study on *wire routing* and *halo type inference* problems. While systematic similarities between the encodings of these problems can be observed in general, ASP is rated more elaboration tolerant on a variation of *wire routing* which includes REACHABILITY constraints. Incidentally, *wire routing* was the problem where CP timed out on all instances. Regarding *halo type inference* problems, Coban et al. point out that presence of recursive definitions and default negation in ASP saves a polynomial number of auxiliary variables. On the other hand, functions and matrices in CP allow to formulate some problems more concisely. Mancini et al. (2008) conducted experiments on *car sequencing*, *word design*, and other CSP that stem from the *CSPLib* (Gent and Walsh, 1999). From their results, it can be observed that there is no a single approach winning on all problem, with ASP being competitive to CP. Experiments confirm, however, that CP benefits from the use of global constraints. Celik et al. (2009) compared ASP and CP on grid puzzles, e.g., *Kakuro*, *Nurikabe*, and *Heyawake*, and draw similar conclusions: The CP approach finds solutions on *Kakuro* instances spending less computational time in comparison to ASP systems, due to the efficient implementation of global constraints. On the other hand, ASP outperforms CP on *Nurikabe* and *Heyawake* due to the unnatural handling of REACHABILITY in CP.

Preliminary work on translating CASP into ASP was conducted by Gebser et al. (2009a) with their CASP system *xpanda*[9]. Their results include a value encoding of the variables' domains and ALL-DIFFERENT using cardinality constraint rules. We have expanded on Gebser et al.'s results in Section 5.1, investigating the level of consistency achieved by ASP inference on the support encoding of ALL-DIFFERENT. Furthermore, we have studied variants of the encoding that maintain bound and range consistency. In our experimental analysis, the effect of *xpanda* is reflected by setting S.

Another related system is *aspartame*[9] (Banbara et al., 2013). Oriented towards prototyping ASP encodings of constraints, it translates a CSP into an ASP specification over externals, i.e., constraint atoms. The latter still need to be implemented, or taken from a library of ASP encodings for variables and constraints, to which our encodings presented in this thesis could serve as a good starting point.

There is also a substantial body of work that focuses on the translation of con-

[9] `http://potassco.sourceforge.net/labs.html`

straints into SAT. In turn, SAT models can be translated into ASP (Niemelä, 1999). Direct encodings into ASP, however, can be more general as every SAT clause can be syntactically represented by a rule, but other ASP constructs are also possible, such as cardinality constraint rules and their generalisation, weight constraint rules (Niemelä et al., 1999).

There are specialised SAT encodings for pseudo-Boolean constraints (Eén and Sörensson, 2006) and integer LINEAR constraints (Tamura et al., 2006). Bessière et al. (2009a) exploit Hall intervals when decomposing ALL-DIFFERENT into cardinality constraints, such that the corresponding propagators achieve bound and range consistency. This result motivated our work on encoding ALL-DIFFERENT into ASP and showing that ASP inference can achieve similar results.

Quimper and Walsh (2006) proposed the GRAMMAR constraint and provided two different domain consistency propagators for the constraint, one based on the CYK parser and one based on the Early parser. Independently, Sellmann (2006) also introduced the GRAMMAR constraint and gave a CYK-based propagator. A SAT encoding of GRAMMAR, based on a decomposition into and-or-graphs, such that UP can prune all values is generated by a CYK-based algorithm was proposed by Quimper and Walsh (2007). We have devised two CYK-based ASP encodings of GRAMMAR in Section 5.2. Our first encoding is fairly straightforward and easily maintainable, and the second is an extension of the first that achieves a result similar to Quimper and Walsh's. Since both approaches compete with the one of Quimper and Walsh in our experimental analysis, we advocate our simpler encoding. Alternatively, by exploiting the fixed length of the constraint, Katsirelos et al. (2009b) gave an algorithm that translates a CFG into an automaton.

DFAs were proposed to specify REGULAR constraints by (Pesant, 2004) who also provided a domain consistency propagator. A DFA-based SAT model of REGULAR such that UP can prune all values was presented by Bacchus (2007). We synthesised this result by providing a DFA-based ASP encoding that achieves a similar result. However, we also presented a simpler, straightforward, and easily maintainable encoding of REGULAR. Given the empirical evidence provided by our GRAMMAR encodings, we also do not expect significant benefits from using the more complex encoding.

Modelling REACHABILITY conditions is folklore in the area of ASP (cf. Niemelä, 1999; Lifschitz, 2008a; Brewka et al., 2011), but the effect of ASP inference has not been previously studied. We have addressed this issue in Section 5.3.

Propagators for a family of CONNECTIVITY constraints, i.e., constraints that en-

sure that a graph is connected, with applications in computational sustainability are presented in (Bessière et al., 2015). For the basic CONNECTIVITY constraint, the propagator enforces domain consistency in linear time.

5.6 Conclusions

In this chapter, we have proceeded with our translation-based approach to constraint answer set solving by introducing more specialised, yet simple encodings for ALL-DIFFERENT, GRAMMAR, REGULAR, and REACHABILITY constraints. In all cases, ASP inference provides an efficient arc, bound, range, or domain consistency propagator for free, i.e., without the need for complex, specialised algorithms, with a similar asymptotic run time complexity.

- Our ASP encodings of ALL-DIFFERENT exploit Hall intervals such that UP can achieve arc, bound, and range consistency, respectively. We have empirically evaluated their performance on CSP benchmarks and found them outperforming hybrid CASP systems and compete with CP solvers.

  The existence of a polynomial-size, domain consistent ASP encoding for ALL-DIFFERENT remains an open problem.

- Our ASP encoding of the GRAMMAR constraint and its restriction to linear grammars and regular grammars is based on the production rules of the grammar. We have also provided an alternative encoding for REGULAR, based on the transition function of a DFA. Extensions to our GRAMMAR and REGULAR encodings sacrifice some of their simplicity but allow the inference of an ASP solver to achieve domain consistency. Though, an experimental analysis has shown only a little advantage over the more straightforward encoding, and both compete with related work. Given the statistical evidence provided by Quimper and Walsh (2008), we expect that our ASP encodings for GRAMMAR beat CP systems, but leave an empirical comparison to future work.

  Open to future analysis is also an encoding of REGULAR constraints based on NFAs rather than DFAs. The encoding would benefit from an NFA being much smaller than DFA. On the other hand, this would break with the one-to-one correspondence between the solutions for the ASP encoding and the ones for the underlying constraint, hence, making solution enumeration more difficult.

– We have analysed the ASP inference on REACHABILITY conditions. In contrast to the intuition that the inference in existing ASP systems naturally handles REACHABILITY, our results have shown that even some restricted variants of REACHABILITY cannot be efficiently propagated by a combination of UP and WFN. This gap can be closed with BL and LD, establishing practical relevance for these forms of inference.

  Recall that, to our knowledge, existing ASP solvers do not implement BL and LD. Therefore, we cannot draw a clear picture from our experimental data, but encourage the integration of BL and LD into ASP systems, e.g., via our flowgraph-based method described earlier in this thesis.

  This thesis investigated the propagation of a folklore ASP encoding of REACHABILITY. Hence, another open problem is the existence of an alternative encoding such that UP alone maintains domain consistency. For instance, a polynomial size, tight encoding might be constructed based on the idea of Warshall's algorithm for transitive closure of directed graphs (Warshall, 1962). This direction will not be investigated in this thesis.

A translation-based approach to constraint answer set solving exhibits many advantages: Since the state of the constraints is made transparent trough translation into a joint encoding, sharing the representation of the variables' domains between encodings, no scheduling of constraint propagation is required. In fact, all constraints are always propagated at the same time by the inference of the underlying ASP solver. The outstanding key advantage of a conflict-driven solver is that CDNL can exploit constraint interdependencies. This can improve propagation between constraints and contribute to the advanced dynamic selection heuristics.

A key limitation of translation-based constraint answer set solving is the worst-case asymptotic space complexity inherent with this strategy. In particular when constraints range over variables with large domains, our encoding can become large, challenging computational resources even if the asymptotic space complexity is a small polynomial. We will address this problem in the next chapter.

Chapter 6

Constraint Answer Set Solving via Lazy Nogood Generation

Our translation-based approach to conflict-driven constraint answer set solving from the previous chapter is made possible by the existence of modern CDNL-based ASP solvers. Their performance, however, is sensitive to the size of ASP encodings which can quickly become impractical. For instance, the best known encoding of ALL-DIFFERENT that achieves domain consistency exploits a worst-case exponential number of Hall sets. But even if the asymptotic space complexity of a constraint's ASP model is a small polynomial, it can challenge computational resources when variables with large domains have to be considered.

This is of no issue for hybrid CASP systems, as they employ a CP solver to handle constraint propagation and variables. On the other hand, hybrid CASP systems sacrifice the advantages of conflict-driven solving within the constraint part of a CASP encoding. This includes conflict-driven learning to uncover constraint interdependencies. In turn, lookback-based heuristics (Mitchell, 2005) are being hindered.

In this chapter we will tackle the disadvantages of both the translation-based and hybrid approaches to constraint answer set solving by combining their advantages.

– We here present a new computational extension to answer set solving, called lazy nogood generation (LNG). Motivated by the success of lazy clause generation (Ohrimenko et al., 2009) in solving CSP, the key idea of LNG is to generate parts of an ASP encoding on demand, only when new information can be propagated.

– We will start with presenting the logical foundations of external propagators by formulating variants of the splitting set theorem (Lifschitz and Turner, 1994). They will allow various alternative characterisations of the conditions to answer sets induced by a (sub-) program (with externals) of the ASP encoding, including unfounded set conditions, completion nogoods, loop nogoods, and external propagators. This allows for programmers to select a representation that best fit their needs.

  In contrast to modern ASP systems like *cmodels* (Giunchiglia et al., 2006), *clasp* (Gebser et al., 2007a), *sup* (Lierler, 2011), and *wasp* (Alviano et al., 2013b), that extract all completion nogoods from an ASP encoding *eagerly*, i.e., during preprocessing, the ones represented by external propagators are not made explicit a-priori. Instead, an external propagator generates parts of its underlying encoding on-demand, in particular, when it triggers any conflict or inference. As we shall see, this technique generalises the idea of encoding the consequences from FL in form of loop nogoods (Lin and Zhao, 2002; Giunchiglia et al., 2006; Gebser et al., 2007a).

– We then show how constraint propagation seamlessly integrates into our framework, and present an algorithmic foundation that is centred around conflict-driven solving. Beyond a CDNL-like decision procedure for ASP solving, it applies LNG via external propagation, resulting in a novel approach to conflict-driven constraint answer set solving. It combines key advantages of hybrid and translation-based approaches, including the exploration of constraint interdependencies.

– Finally, we empirically evaluate our approach and compare to the state-of-the-art in ASP and CASP.

We have implemented a prototypical CASP system to demonstrate some key principles of our approaches. In 2013, it has successfully participated in the fourth ASP competition (Alviano et al., 2013a), outperforming hybrid systems.

6.1 Nogoods of Programs with Externals

We start with a generalisation of the approach of Gebser et al. (2007a), i.e., describing nogoods that capture the inferences from the completion of a program Π with externals $\mathscr{E}$.

Definition 6.1: Completion Nogoods of a Program with Externals

For a program Π over $\mathscr{P}$ with externals $\mathscr{E}$, the set of *completion nogoods*, denoted by $\Delta_\Pi^{\mathscr{E}}$, is the following:

$$\Delta_\Pi^{\mathscr{E}} = \bigcup_{\beta \in \mathrm{body}(\Pi)} \mathrm{EQ}_\beta \cup \bigcup_{p \in \mathscr{P} \setminus \mathscr{E}} \mathbf{\Delta}_\Pi^p$$

The sets EQ_β and $\mathbf{\Delta}_\Pi^p$ are the same as defined in the Background chapter of this thesis. It is easy to see that if there are no externals, say $\mathscr{E} = \emptyset$, then $\Delta_\Pi^\emptyset$ represents precisely the set of completion nogoods of Π in the style of Gebser et al., i.e., $\Delta_\Pi^\emptyset = \Delta_\Pi$. Hence, by Theorem 2.2, the solutions for $\Delta_\Pi^\emptyset$ correspond to the supported models of Π. In general, however, $\Delta_\Pi^{\mathscr{E}}$ excludes $\mathbf{\Delta}_\Pi^p$ for each external atom p, as external atoms are naturally defined outside of Π. (Observe that, otherwise, for each $p \in \mathscr{E}$ we would have $\mathbf{\Delta}_\Pi^p = \{\{\mathbf{T}p\}\}$ since $\mathrm{ES}_\Pi(\{p\}) = \emptyset$, i.e., there is no rule $r \in \Pi$ with $\mathrm{head}(r) = p$, unintentionally excluding p from any solution.) The following proposition establishes correspondance with Gebser et al.'s completion nogoods of a program.

Lemma 6.1: Splitting Completion Nogoods

Let Π be a program over $\mathscr{P}$ with externals $\mathscr{F}$, and $\mathscr{E}$ a splitting set for Π. Then, we have that the equation $\Delta_\Pi^{\mathscr{F}} = \Delta_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}^{\mathscr{E}\cup\mathscr{F}} \cup \Delta_{\Pi_{\mathscr{E}}}^{\mathscr{F}}$ holds. In particular, if $\mathscr{F} = \emptyset$ then $\Delta_\Pi = \Delta_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}^{\mathscr{E}} \cup \Delta_{\Pi_{\mathscr{E}}}$.

Another direct consequence of the lemma is that for an assignment $\mathbf{A}$, we have that $\mathbf{A}$ is a solution for Δ_Π if and only if $\mathbf{A}$ is a conflict-free assignment for $\Delta_{\Pi_{\mathscr{E}}}$ and $\Delta_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Such assignment $\mathbf{A}$, however, is technically not a solution for $\Delta_{\Pi_{\mathscr{E}}}$ and $\Delta_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ because $\mathrm{dom}(\mathbf{A})$ goes beyond the atoms and bodies of $\Pi_{\mathscr{E}}$ and $\Pi_{\mathscr{P}\setminus\mathscr{E}}$, respectively. To restrict the domain of an assignment to the objects in a program, we define

$$\mathbf{A}|_\Pi = \{\sigma | \sigma \in \mathbf{A},\ \mathrm{var}(\sigma) \in \mathrm{atom}(\Pi) \cup \mathrm{body}(\Pi)\}.$$

Then, we have that $\mathbf{A}$ is a solution for Δ_Π if and only if $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is a solution for $\Delta_{\Pi_{\mathscr{E}}}$ and $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$.

Proof of Lemma 6.1. We start by applying Definition 6.1.

$$\begin{aligned}\Delta_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}^{\mathscr{E}\cup\mathscr{F}} \cup \Delta_{\Pi_{\mathscr{E}}}^{\mathscr{F}} = &\bigcup_{\beta \in \mathrm{body}(\Pi_{\mathscr{P}\setminus\mathscr{E}})} \mathrm{EQ}_\beta \cup \bigcup_{p \in \mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F})} \mathbf{\Delta}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}^p \\ &\cup \bigcup_{\beta \in \mathrm{body}(\Pi_{\mathscr{E}})} \mathrm{EQ}_\beta \cup \bigcup_{p \in \mathscr{E}\setminus\mathscr{F}} \mathbf{\Delta}_{\Pi_{\mathscr{E}}}^p\end{aligned}$$

Since $\mathscr{E}$ is a splitting set for Π, for every $p \in \mathscr{P} \setminus (\mathscr{E} \cup \mathscr{F})$ we have that $\mathrm{ES}_\Pi(\{p\}) \cap \mathrm{body}(\Pi_\mathscr{E}) = \emptyset$, and for every $p \in \mathscr{E} \setminus \mathscr{F}$ we have that $\mathrm{ES}_\Pi(\{p\}) \cap \mathrm{body}(\Pi_{\mathscr{P}\setminus\mathscr{E}}) = \emptyset$. In other words, the rules in $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ never share a head with rules in $\Pi_\mathscr{E}$, and vice-versa. Hence, for every $p \in \mathscr{P} \setminus (\mathscr{E} \cup \mathscr{F})$ it holds that $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{p\}) = \mathrm{ES}_\Pi(\{p\})$, and for every $p \in \mathscr{E} \setminus \mathscr{F}$ it holds that $\mathrm{ES}_{\Pi_\mathscr{E}}(\{p\}) = \mathrm{ES}_\Pi(\{p\})$. We proceed with applying these equations, and get the following:

$$
\begin{aligned}
&= \bigcup_{\beta \in \mathrm{body}(\Pi_{\mathscr{P}\setminus\mathscr{E}})} \mathrm{EQ}_\beta \cup \bigcup_{p \in \mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F})} \mathbf{\Delta}^p_\Pi \\
&\quad \cup \bigcup_{\beta \in \mathrm{body}(\Pi_\mathscr{E})} \mathrm{EQ}_\beta \cup \bigcup_{p \in \mathscr{E}\setminus\mathscr{F}} \mathbf{\Delta}^p_\Pi \\
&= \bigcup_{\beta \in \mathrm{body}(\Pi_{\mathscr{P}\setminus\mathscr{E}}) \cup \mathrm{body}(\Pi_\mathscr{E})} \mathrm{EQ}_\beta \cup \bigcup_{p \in (\mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F})) \cup (\mathscr{E}\setminus\mathscr{F})} \mathbf{\Delta}^p_\Pi \\
&= \bigcup_{\beta \in \mathrm{body}(\Pi)} \mathrm{EQ}_\beta \cup \bigcup_{p \in \mathscr{P}\setminus\mathscr{F}} \mathbf{\Delta}^p_\Pi \\
&= \Delta^\mathscr{F}_\Pi
\end{aligned}
$$

This proves the equation. □

We will take up a previous example to illustrate the lemma. It will also serve as a running example.

Example 6.1

Reconsider the splitting set $\mathscr{E} = \{p, q\}$ for the program Π over $\mathscr{P} = \{p, q, r, s\}$ from Example 2.19, given through

$$
\Pi_\mathscr{E} = \left\{ \begin{array}{ll} r_1: & p \leftarrow not\ q \\ r_2: & q \leftarrow not\ p \end{array} \right\} \text{ and } \Pi_{\mathscr{P}\setminus\mathscr{E}} = \left\{ \begin{array}{ll} r_3: & r \leftarrow p \\ r_4: & r \leftarrow s \\ r_5: & s \leftarrow r \end{array} \right\}.
$$

Verify that the completion nogoods represented by $\Pi_\mathscr{E}$ are

$$
\begin{aligned}
\Delta^\emptyset_{\Pi_\mathscr{E}} &= \mathrm{EQ}_{\mathrm{body}(r_1)} \cup \mathrm{EQ}_{\mathrm{body}(r_2)} \cup \mathbf{\Delta}^p_{\Pi_\mathscr{E}} \cup \mathbf{\Delta}^q_{\Pi_\mathscr{E}} \\
&= \left\{ \begin{array}{l} \{\mathbf{F}\mathrm{body}(r_1), \mathbf{F}q\}, \{\mathbf{T}\mathrm{body}(r_1), \mathbf{T}q\}, \\ \{\mathbf{F}\mathrm{body}(r_2), \mathbf{F}p\}, \{\mathbf{T}\mathrm{body}(r_2), \mathbf{T}p\}, \\ \{\mathbf{T}p, \mathbf{F}\mathrm{body}(r_1)\}, \{\mathbf{F}p, \mathbf{T}\mathrm{body}(r_1)\}, \\ \{\mathbf{T}q, \mathbf{F}\mathrm{body}(r_2)\}, \{\mathbf{F}q, \mathbf{T}\mathrm{body}(r_2)\} \end{array} \right\},
\end{aligned}
$$

the ones represented by $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ are

$$\begin{aligned}\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} &= \mathrm{EQ}_{\mathrm{body}(r_3)} \cup \cdots \cup \mathrm{EQ}_{\mathrm{body}(r_5)} \cup \boldsymbol{\Delta}^{r}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \boldsymbol{\Delta}^{s}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \\ &= \left\{ \begin{array}{l} \{\mathbf{F}\,\mathrm{body}(r_3), \mathbf{T}p\}, \{\mathbf{T}\,\mathrm{body}(r_3), \mathbf{F}p\}, \\ \{\mathbf{F}\,\mathrm{body}(r_4), \mathbf{T}s\}, \{\mathbf{T}\,\mathrm{body}(r_4), \mathbf{F}s\}, \\ \{\mathbf{F}\,\mathrm{body}(r_5), \mathbf{T}r\}, \{\mathbf{T}\,\mathrm{body}(r_5), \mathbf{F}r\}, \\ \{\mathbf{T}r, \mathbf{F}\,\mathrm{body}(r_3), \mathbf{F}\,\mathrm{body}(r_4)\}, \\ \{\mathbf{F}r, \mathbf{T}\,\mathrm{body}(r_3)\}, \{\mathbf{F}r, \mathbf{T}\,\mathrm{body}(r_4)\}, \\ \{\mathbf{T}s, \mathbf{F}\,\mathrm{body}(r_5)\}, \{\mathbf{F}s, \mathbf{T}\,\mathrm{body}(r_5)\} \end{array} \right\},\end{aligned}$$

and verify that $\Delta^{\emptyset}_{\Pi_{\mathscr{E}}} \cup \Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \Delta_{\Pi}$, as detailled in Example 2.12.

As with completion nogoods, we can generalise the notion of unfounded sets for programs with externals. They are similar to unfounded sets of a program, but do not include externals.

Definition 6.2: Unfounded Set of a Program with Externals

Let Π be a program with externals $\mathscr{E}$. Given an assignment $\mathbf{A}$, $U \subseteq \mathrm{atom}(\Pi)$ is an *unfounded set* of Π w.r.t. $\mathbf{A}$ if $U \cap \mathscr{E} = \emptyset$ and $\mathrm{ES}_{\Pi}(U) \subseteq \mathbf{A}^{\mathbf{F}}$.

It is easy to see that if the set of externals is empty, say $\mathscr{E} = \emptyset$, then the unfounded sets of Π coincide with the conventional ones. The following proposition guarantees that, for any assignment, the process of splitting a logic program maintains unfounded-freeness.

Lemma 6.2

Let Π be a program over $\mathscr{P}$, $\mathscr{E}$ a splitting set for Π, and $\mathbf{A}$ an assignment. Then, we have that $\mathbf{A}$ is unfounded-free for Π if and only if $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{E}}$ and $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$.

Proof. We show both implications of the proposition.

($\Rightarrow$) Let $\mathbf{A}$ be unfounded-free for Π. First, we show that $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{E}}$, i.e., for every $U \subseteq \mathrm{atom}(\Pi_{\mathscr{E}})$ such that $\mathrm{ES}_{\Pi_{\mathscr{E}}}(U) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{E}}} = \emptyset$ we have that $U \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{E}}}$. To begin, let $U \subseteq \mathrm{atom}(\Pi_{\mathscr{E}})$ such that $\mathrm{ES}_{\Pi_{\mathscr{E}}}(U) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{E}}} = \emptyset$. By definition, for every $r \in \Pi_{\mathscr{P}\setminus\mathscr{E}}$ we have that $\mathrm{head}(r) \notin \mathscr{E}$. Hence, since $U \subseteq \mathrm{atom}(\Pi_{\mathscr{E}}) \subseteq \mathscr{E}$, we have $\mathrm{ES}_{\Pi_{\mathscr{E}}}(U) = \mathrm{ES}_{\Pi}(U)$, and therefore, $\mathrm{ES}_{\Pi}(U) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{E}}} = \emptyset$. Since $\mathbf{A}|_{\Pi_{\mathscr{E}}} \subseteq \mathbf{A}$ and $\mathbf{A}$ is unfounded-free for Π, we also have $U \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{E}}}$. In conclusion, $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{E}}$.

Second, we show that $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$, i.e., for every $U \subseteq$

atom($\Pi_{\mathscr{P}\setminus\mathscr{E}}$) such that $U \cap \mathscr{E} = \emptyset$ and $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \emptyset$ we have that $U \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. To begin, let $U \subseteq \mathrm{atom}(\Pi_{\mathscr{P}\setminus\mathscr{E}})$ such that $U \cap \mathscr{E} = \emptyset$ and $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \emptyset$. By definition, for every $r \in \Pi_{\mathscr{E}}$ we have that $\mathrm{head}(r) \notin \mathrm{atom}(\Pi_{\mathscr{P}\setminus\mathscr{E}}) \setminus \mathscr{E}$. Hence, since $U \subseteq \mathrm{atom}(\Pi_{\mathscr{P}\setminus\mathscr{E}}) \setminus \mathscr{E}$, we have $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) = \mathrm{ES}_{\Pi}(U)$, and therefore, $\mathrm{ES}_{\Pi}(U) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \emptyset$. Since $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \subseteq \mathbf{A}$ and $\mathbf{A}$ is unfounded-free for Π, we also have $U \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. In conclusion, $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$.

($\Leftarrow$) Let $\mathbf{A}$ be an assignment such that $\mathbf{A}|_{\Pi_{\mathscr{P}}}$ is unfounded-free for $\Pi_{\mathscr{E}}$ and $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$. We show that $\mathbf{A}$ is unfounded-free for Π, i.e., for every $U \subseteq \mathscr{P}$ such that $\mathrm{ES}_{\Pi}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset$ we have that $U \subseteq \mathbf{A}^{\mathbf{F}}$. To begin, let $U \subseteq \mathscr{P}$ such that $\mathrm{ES}_{\Pi}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset$. Recall that, by definition, for every $r \in \Pi_{\mathscr{E}}$ we have that $\mathrm{head}(r) \in \mathscr{E}$, and for every rule $r \in \Pi_{\mathscr{P}\setminus\mathscr{E}}$ we have $\mathrm{head}(r) \in \mathscr{P} \setminus \mathscr{E}$. Hence, we have $\mathrm{ES}_{\Pi}(U) = \mathrm{ES}_{\Pi_{\mathscr{E}}}(U) \cup \mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U)$. Given this, from $\mathrm{ES}_{\Pi}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset$ we infer $\mathrm{ES}_{\Pi_{\mathscr{E}}}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset$ and $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset$. In particular, we have $\mathrm{ES}_{\Pi_{\mathscr{E}}}(U \cap \mathscr{E}) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}}} = \emptyset$, and $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U \setminus \mathscr{E}) \setminus \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \emptyset$. Since $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{E}}$ and $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$, we have $U \cap \mathscr{E} \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{E}}}$ and $U \setminus \mathscr{E} \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Moreover, since $\mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{E}}} \cup \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \subseteq \mathbf{A}^{\mathbf{F}}$, we get $U \subseteq \mathbf{A}^{\mathbf{F}}$. In conclusion, $\mathbf{A}$ is unfounded-free for Π. □

Example 6.2

We proceed from Example 6.1. Consider the assignments $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ given through

$$\mathbf{A}_1 = \{\mathbf{T}p, \mathbf{F}q, \mathbf{T}r, \mathbf{T}s, \mathbf{T}\mathrm{body}(r_1), \mathbf{F}\mathrm{body}(r_2), \mathbf{T}\mathrm{body}(r_3), \mathbf{T}\mathrm{body}(r_4), \mathbf{T}\mathrm{body}(r_5)\}$$
$$\mathbf{A}_2 = \{\mathbf{F}p, \mathbf{T}q, \mathbf{T}r, \mathbf{T}s, \mathbf{F}\mathrm{body}(r_1), \mathbf{T}\mathrm{body}(r_2), \mathbf{F}\mathrm{body}(r_3), \mathbf{T}\mathrm{body}(r_4), \mathbf{T}\mathrm{body}(r_5)\}$$
$$\mathbf{A}_3 = \{\mathbf{F}p, \mathbf{T}q, \mathbf{F}r, \mathbf{F}s, \mathbf{F}\mathrm{body}(r_1), \mathbf{T}\mathrm{body}(r_2), \mathbf{F}\mathrm{body}(r_3), \mathbf{F}\mathrm{body}(r_4), \mathbf{F}\mathrm{body}(r_5)\}$$

Verify that $\mathbf{A}_1$ and $\mathbf{A}_3$ are unfounded-free for Π, $\mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ are unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$, and $\mathbf{A}_1|_{\Pi_{\mathscr{E}}}$ and $\mathbf{A}_3|_{\Pi_{\mathscr{E}}}$ are unfounded-free for $\Pi_{\mathscr{E}}$, respectively. On the other hand, for $U = \{r, s\}$ we have $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) = \mathrm{ES}_{\Pi}(U) = \{\mathrm{body}(r_3)\}$ where $\mathrm{body}(r_3) \in \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \subseteq \mathbf{A}^{\mathbf{F}}_2$. That is, U is an unfounded set for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and for Π w.r.t. $\mathbf{A}_2$.

In light of Lemmas 6.1 and 6.2, we formulate a variant of the splitting set theorem (Lifschitz and Turner, 1994).

Recall, the splitting set theorem provides conditions under which the evaluation of a program can be split into the evaluation of (sub-) programs, e.g., for a program Π and a splitting set $\mathscr{E}$ for Π, splitting Π into the program $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ with ex-

ternals $\mathscr{E}$ and the program $\Pi_{\mathscr{P}}$ (without externals). Applied recursively, the splitting set theorem allows for the application of a different representation, or solving technique, for each (sub-) program of an ASP encoding.

Intuitively, the following variant states that, given a splitting set $\mathscr{E}$, we can compute the answer sets of a program Π by computing the answer sets of $\Pi_{\mathscr{E}}$ and unfounded-free solutions for the completion nogoods of $\Pi_{\mathscr{P}\setminus\mathscr{E}}$.

Theorem 6.3

Let Π be a program over $\mathscr{P}$, $\mathscr{E}$ a splitting set for Π, $X \subseteq \mathscr{P}$ and $\mathbf{A}$ the assignment given through

$$\begin{aligned}\mathbf{A} = &\{\mathbf{T}p \mid p \in X\} \cup \{\mathbf{F}p \mid p \in \mathscr{P} \setminus X\}\\ &\cup\{\mathbf{T}\,\mathrm{body}(r) \mid r \in \Pi, \mathrm{body}(r)^+ \subseteq X, \mathrm{body}(r)^- \cap X = \emptyset\}\\ &\cup\{\mathbf{F}\,\mathrm{body}(r) \mid r \in \Pi, (\mathrm{body}(r)^+ \cap (\mathscr{P} \setminus X))\\ &\qquad\qquad \cup (\mathrm{body}(r)^- \cap X) \neq \emptyset\}.\end{aligned}$$

Then, X is an answer set of Π if and only if

- $X \cap \mathscr{E}$ is an answer set of $\Pi_{\mathscr{E}}$, and
- $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$.

When applied recursively, a direct consequence of the theorem is that the conditions on the answer sets induced by any (sub-) program (over externals) can be represented by the unfounded-free solutions to its completion nogoods.

Proof of Theorem 6.3. We show both implications of the proposition.

($\Rightarrow$) Let X be an answer set of Π. By Theorem 2.3, the assignment $\mathbf{A}$ is a solution for Δ_Π and unfounded-free for Π. Then, by Lemma 6.1, $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is a solution for $\Delta^{\emptyset}_{\Pi_{\mathscr{E}}}$ and $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, and by Lemma 6.2, $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{E}}$ and $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$. Since $\Delta^{\emptyset}_{\Pi_{\mathscr{E}}} = \Delta_{\Pi_{\mathscr{E}}}$ and $\mathscr{E} = \mathrm{atom}(\Pi_{\mathscr{E}})$, again by Theorem 2.3, $\mathbf{A}|^{\mathbf{T}}_{\Pi_{\mathscr{E}}} \cap \mathscr{E}$ is an answer set of $\Pi_{\mathscr{E}}$. In particular, since $X = \mathbf{A}|^{\mathbf{T}}_{\Pi_{\mathscr{E}}} \cap \mathscr{P}$ and $\mathscr{E} \subseteq \mathscr{P}$, $X \cap \mathscr{E}$ is an answer set of $\Pi_{\mathscr{E}}$.

($\Leftarrow$) Let $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ be a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$, and $X \cap \mathscr{E}$ be an answer set of $\Pi_{\mathscr{E}}$. Then, by Theorem 2.3, the assignment $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is a solution for $\Delta_{\Pi_{\mathscr{E}}}$ and unfounded-free for $\Pi_{\mathscr{E}}$. Since $\Delta_{\Pi_{\mathscr{E}}} = \Delta^{\emptyset}_{\Pi_{\mathscr{E}}}$, we have that $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is also a solution for $\Delta^{\emptyset}_{\Pi_{\mathscr{E}}}$ and unfounded-free for $\Pi_{\mathscr{E}}$. Then, by Lemma 6.1, $\mathbf{A}$ is a solution for Δ_Π, and by Lemma 6.2, $\mathbf{A}$ is unfounded-free for Π. Finally, by Theorem 2.3, $X = \mathbf{A}^{\mathbf{T}} \cap \mathscr{P}$ is an answer set of Π. $\square$

Example 6.3

We proceed from Example 6.2. Verify that the assignments $\mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, and $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ are solutions for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Consider the projections of the underlying assignments onto the atoms in $\mathscr{P}$ given through

$$X_1 = \mathbf{A}_1 \cap \mathscr{P} = \{p, r, s\},$$
$$X_2 = \mathbf{A}_2 \cap \mathscr{P} = \{q, r, s\}, \text{ and}$$
$$X_3 = \mathbf{A}_3 \cap \mathscr{P} = \{q\}.$$

Verify that $(X_1 \cap \mathscr{E}) = \{p\}$ and $(X_2 \cap \mathscr{E}) = (X_3 \cap \mathscr{E}) = \{q\}$ are answer sets of $\Pi_{\mathscr{E}}$.

However, as we have demonstrated in Example 6.2, the assignment $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is not unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$. Therefore, by Theorem 6.3, only sets X_1 and X_3 are answer sets of Π. Indeed, they are (cf. Example 2.11).

As with conventional unfounded sets, we can capture the conditions induced by unfounded sets of a program with externals in terms of nogoods.

Definition 6.3: Loop Nogoods of a Program with Externals

Let Π be a program over $\mathscr{P}$ with externals $\mathscr{E}$. The set of *loop nogoods* of Π, denoted by $\Lambda^{\mathscr{E}}_{\Pi}$, is the following.

$$\Lambda^{\mathscr{E}}_{\Pi} = \bigcup_{U \subseteq \mathscr{P}\setminus\mathscr{E}} \{\lambda^{p}_{\mathrm{ES}_{\Pi}(U)} \mid p \in U\}$$

The nogood $\lambda^{p}_{\mathrm{ES}_{\Pi}(U)}$ is the same as defined in the Background chapter of this thesis. It is easy to see that if there are no externals, say $\mathscr{E} = \emptyset$, then $\Lambda^{\emptyset}_{\Pi} = \Lambda_{\Pi}$. In general, however, $\Lambda^{\mathscr{E}}_{\Pi}$ excludes $\lambda^{p}_{\mathrm{ES}_{\Pi}(U)}$ for sets U which contain externals, i.e., we have $U \cap \mathscr{E} = \emptyset$.

Lemma 6.4: Splitting Loop Nogoods

Let Π be a program over $\mathscr{P}$ with externals $\mathscr{F}$, and $\mathscr{E}$ a splitting set for Π. Then, we have that the equation $\Lambda^{\mathscr{F}}_{\Pi} \supseteq \Lambda^{\mathscr{E}\cup\mathscr{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{F}}_{\Pi_{\mathscr{E}}}$ holds. In particular, if $\mathscr{F} = \emptyset$ then $\Lambda_{\Pi} \supseteq \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda_{\Pi_{\mathscr{E}}}$.

Observe that some loop nogoods of the original program can be lost through splitting, that are, loop nogoods for sets of atoms that receive external support from both sub-programs. However, as we shall see, those nogoods are not required to establish correspondence between solutions and answer sets.

Proof of Lemma 6.4. We start by applying Definition 6.3.

$$\Lambda^{\mathscr{E}\cup\mathscr{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{F}}_{\Pi_{\mathscr{E}}} = \bigcup_{U\subseteq\mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F})}\{\lambda^{p}_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U)} \mid p\in U\} \cup \bigcup_{U\subseteq\mathscr{E}\setminus\mathscr{F}}\{\lambda^{p}_{\mathrm{ES}_{\Pi_{\mathscr{E}}}(U)} \mid p\in U\}$$

Since $\mathscr{E}$ is a splitting set for Π, for every $U \subseteq \mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F})$ we have that $\mathrm{ES}_\Pi(U)\cap \mathrm{body}(\Pi_{\mathscr{E}}) = \emptyset$, and for every $U\subseteq\mathscr{E}\setminus\mathscr{F}$ we have that $\mathrm{ES}_\Pi(U)\cap\mathrm{body}(\Pi_{\mathscr{P}\setminus\mathscr{E}}) = \emptyset$. In other words, the rules in $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ never share a head with rules in $\Pi_{\mathscr{E}}$, and vice-versa. Hence, for every $U\subseteq\mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F})$ it holds that $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) = \mathrm{ES}_\Pi(U)$, and for every $U\subseteq\mathscr{E}\setminus\mathscr{F}$ it holds that $\mathrm{ES}_{\Pi_{\mathscr{E}}}(U) = \mathrm{ES}_\Pi(U)$. We proceed with applying these equations, and get the following:

$$\begin{aligned}
&= \bigcup_{U\subseteq\mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F})}\{\lambda^{p}_{\mathrm{ES}_\Pi(U)} \mid p\in U\} \cup \bigcup_{U\subseteq\mathscr{E}\setminus\mathscr{F}}\{\lambda^{p}_{\mathrm{ES}_\Pi(U)} \mid p\in U\}\\
&\subseteq \bigcup_{U\subseteq(\mathscr{P}\setminus(\mathscr{E}\cup\mathscr{F}))\cup(\mathscr{E}\setminus\mathscr{F})}\{\lambda^{p}_{\mathrm{ES}_\Pi(U)} \mid p\in U\}\\
&= \bigcup_{U\subseteq((\mathscr{P}\setminus\mathscr{E})\setminus\mathscr{F})\cup(\mathscr{E}\setminus\mathscr{F})}\{\lambda^{p}_{\mathrm{ES}_\Pi(U)} \mid p\in U\}\\
&= \bigcup_{U\subseteq\mathscr{P}\setminus\mathscr{F}}\{\lambda^{p}_{\mathrm{ES}_\Pi(U)} \mid p\in U\}\\
&= \Lambda^{\mathscr{F}}_{\Pi}
\end{aligned}$$

This proves the inclusion. Observe, however, that equality holds if we only consider sets $U\subseteq\mathrm{loop}(\Pi)$. □

The following proposition establishes, for any assignment, the correspondence between unfounded sets and loop nogoods included in the assignment for programs with externals.

Lemma 6.5

Let Π be a program with externals $\mathscr{E}$, and $\mathbf{A}$ an assignment. Then, U is an unfounded set of Π w.r.t. $\mathbf{A}$ such that $U\cap\mathbf{A^T}\neq\emptyset$ if and only if there is some $\delta\in\Lambda^{\mathscr{E}}_{\Pi}$ such that $\delta\subseteq\mathbf{A}$.

Proof. The proof follows from Definitions 6.2 and 6.3. We show both implications of the proposition.

($\Rightarrow$) Let U be an unfounded set of Π w.r.t. $\mathbf{A}$ and $p\in U\cap\mathbf{A^T}$. For $\mathrm{ES}_\Pi(U) = \{\beta_1,\ldots,\beta_k\}$ we have, by Definition 6.2, $U\cap\mathscr{E}=\emptyset$ and $\{\beta_1,\ldots,\beta_k\}\subseteq\mathbf{A^F}$. Hence, we have $\delta = \lambda^{p}_{\mathrm{ES}_\Pi(U)} = \{\mathbf{T}p,\mathbf{F}\beta_1,\ldots,\mathbf{F}\beta_k\}\in\Lambda^{\mathscr{E}}_{\Pi}$ such that $\delta\subseteq\mathbf{A}$.

($\Leftarrow$) Let $\delta\in\Lambda^{\mathscr{E}}_{\Pi}$ be a loop nogood such that $\delta\subseteq\mathbf{A}$. Since δ is of the form $\lambda^{p}_{\mathrm{ES}_\Pi(U)} = \{\mathbf{T}p,\mathbf{F}\beta_1,\ldots,\mathbf{F}\beta_k\}$ where $p\in U\subseteq\mathrm{atom}(\Pi)\setminus\mathscr{E}$ and $\mathrm{ES}_\Pi(U)=\{\beta_1,\ldots,\beta_k\}$, we have that there is $p\in U\cap\mathbf{A^T}$ where $U\cap\mathscr{E}=\emptyset$ and $\mathrm{ES}_\Pi(U)\setminus\mathbf{A^F}=\emptyset$. Hence, by Definition 6.2, U is an unfounded set of Π w.r.t. $\mathbf{A}$ and $p\in U\cap\mathbf{A^T}$. □

Example 6.4

We proceed from Example 6.2. Recall that $\{r, s\}$ is an unfounded set for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, respectively. However, $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$, whilst $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is not. We now inspect the set of nogoods $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, given through

$$\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \{\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r\})}, \lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{s\})}, \lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}, \lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}\}$$

where

$$\begin{aligned}
\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r\})} &= \{\mathbf{T}r, \mathbf{F}\mathrm{body}(r_3), \mathbf{F}\mathrm{body}(r_4)\},\\
\lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r\})} &= \{\mathbf{T}s, \mathbf{F}\mathrm{body}(r_5)\},\\
\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} &= \{\mathbf{T}r, \mathbf{F}\mathrm{body}(r_3)\}, \text{ and}\\
\lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} &= \{\mathbf{T}s, \mathbf{F}\mathrm{body}(r_3)\}.
\end{aligned}$$

Actually, since $\mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, and $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ are solutions for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, we only inspect the nogoods $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \setminus \Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \{\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}, \lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}\}$, as for all the other nogoods δ we already know that $\delta \not\subseteq \mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, $\delta \not\subseteq \mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, and that $\delta \not\subseteq \mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$.

To begin with, verify that $\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} \subseteq \mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} \subseteq \mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Therefore, by Lemma 6.5, $\{r, s\}$ is an unfounded set of $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ that contains true atoms. Indeed, $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is not unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ (cf. Example 6.2).

On the other hand, verify that $\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} \not\subseteq \mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} \not\subseteq \mathbf{A}_3$, i.e., $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ does not contain any nogood from $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Therefore, by Lemma 6.5, there is no unfounded set of $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ that contains true atoms. Indeed, $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$.

At last, recall that there is no unfounded set of $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. $\mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Hence, by Lemma 6.5, $\delta \not\subseteq \mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ for every $\delta \in \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Indeed, verify that we have $\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} \not\subseteq \mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})} \not\subseteq \mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$.

Based on Lemma 6.5, we formulate another variant of the splitting set theorem. It states that, given a splitting set $\mathscr{E}$, we can compute the answer sets of a program Π by computing the answer sets of $\Pi_{\mathscr{E}}$ and solutions for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$.

Theorem 6.6

Let Π be a program over $\mathscr{P}$, $\mathscr{E}$ a splitting set for Π, $X \subseteq \mathscr{P}$ and $\mathbf{A}$ the assignment

given through

$$\begin{aligned}\mathbf{A} = &\{\mathbf{T}p \mid p \in X\} \cup \{\mathbf{F}p \mid p \in \mathscr{P} \setminus X\}\\ &\cup\{\mathbf{T}\,\mathrm{body}(r) \mid r \in \Pi, \mathrm{body}(r)^+ \subseteq X, \mathrm{body}(r)^- \cap X = \emptyset\}\\ &\cup\{\mathbf{F}\,\mathrm{body}(r) \mid r \in \Pi, (\mathrm{body}(r)^+ \cap (\mathscr{P} \setminus X))\\ &\qquad\qquad \cup (\mathrm{body}(r)^- \cap X) \neq \emptyset\}.\end{aligned}$$

Then, X is an answer set of Π if and only if

- $X \cap \mathscr{E}$ is an answer set of $\Pi_{\mathscr{E}}$, and
- $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$.

Proof. The proof follows that of Theorem 6.3, but applies Lemma 6.5. We show both implications of the proposition.

($\Rightarrow$) Let X be an answer set of Π. By Theorem 6.3, $X \cap \mathscr{E}$ is an answer set of $\Pi_{\mathscr{E}}$, and the assignment $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$. In particular, for every $U \subseteq \mathscr{P} \setminus \mathscr{E}$ such that $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset$ we have $U \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Then, by Lemma 6.5, we also have for every $\delta \in \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ that $\delta \not\subseteq \mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. In other words, $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Hence, $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$.

($\Leftarrow$) Let $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ be a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, and $X \cap \mathscr{E}$ be an answer set of $\Pi_{\mathscr{E}}$. Then, by Theorem 2.3, the assignment $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is a solution for $\Delta_{\Pi_{\mathscr{E}}}$ and unfounded-free for $\Pi_{\mathscr{E}}$. Since $\Delta_{\Pi_{\mathscr{E}}} = \Delta^{\emptyset}_{\Pi_{\mathscr{E}}}$, we have that $\mathbf{A}|_{\Pi_{\mathscr{E}}}$ is also a solution for $\Delta^{\emptyset}_{\Pi_{\mathscr{E}}}$ and unfounded-free for $\Pi_{\mathscr{E}}$. On the other hand, from $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ being a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, by Lemma 6.5, we have for every unfounded set U of $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ that $U \cap \mathbf{A}|^{\mathbf{T}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} = \emptyset$. In fact, since $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is total, for every unfounded set U of $\Pi_{\mathscr{P}\setminus\mathscr{E}}$ w.r.t. $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ we have $U \subseteq \mathbf{A}|^{\mathbf{F}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Hence, $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is unfounded-free for $\Pi_{\mathscr{P}\setminus\mathscr{E}}$. Then, by Lemma 6.1, $\mathbf{A}$ is a solution for Δ_{Π}, and by Lemma 6.2, $\mathbf{A}$ is unfounded-free for Π. Finally, by Theorem 2.3, $X = \mathbf{A}^{\mathbf{T}} \cap \mathscr{P}$ is an answer set of Π. □

Example 6.5

We proceed from Example 6.4. Recall that $\mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ are solutions for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, whilst $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ but not for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Moreover, recall that for $X_1 = \mathbf{A}_1 \cap \mathscr{P} = \{p, r, s\}$, $X_2 = \mathbf{A}_2 \cap \mathscr{P} = \{q, r, s\}$, and $X_3 = \mathbf{A}_3 \cap \mathscr{P} = \{q\}$ we have that $(X_1 \cap \mathscr{E}) = \{p\}$, and $(X_2 \cap \mathscr{E}) = (X_3 \cap \mathscr{E}) = \{q\}$ are an answer sets of $\Pi_{\mathscr{E}}$. Therefore, by Theorem 6.6, only sets X_1 and X_3 are answer sets of Π. Indeed, they are (cf. Example 2.11).

6.2 Lazy Nogood Generation

Given a program Π, instead of generating all nogoods Δ_Π (or even Λ_Π) a-priori for use in UP, referred to as *eager* encoding, we introduce external propagators to generate nogoods on demand, i.e., only when UP is able to propagate new information. We call this technique lazy nogood generation (LNG).

Definition 6.4: External Propagator

An *external propagator* for a set of nogoods Γ is a function p that maps a Boolean assignment to a subset of Γ such that for any total assignment $\mathbf{A}$ if $\delta \subseteq \mathbf{A}$ for some $\delta \in \Gamma$ then $\delta' \subseteq \mathbf{A}$ for some $\delta' \in \mathsf{p}(\mathbf{A}) \subseteq \Gamma$.

In other words, an external propagator generates a conflicting nogood from Γ if some nogood in Γ is conflicting with a total assignment. If this condition holds for every (partial) assignment, then we call the external propagator *conflict-optimal*.

Example 6.6

Consider the external propagator p_1 for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, given through

$$\mathsf{p}_1(\mathbf{A}) = \begin{cases} \{\lambda^p_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U)} \mid U \in \mathrm{loop}(\Pi_{\mathscr{P}\setminus\mathscr{E}}), p \in U \cap \mathbf{A}^{\mathbf{T}}, \mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset\} & \text{if } \mathbf{A} \text{ is total,} \\ \emptyset & \text{else.} \end{cases}$$

It is easy to verify that p_1 is an external propagator for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. In fact, every loop nogood that p_1 generates has the form $\delta = \{\lambda^p_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U)}\} = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$ where $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) = \{\beta_1, \ldots, \beta_k\}$. Since $p \in \mathbf{A}^{\mathbf{T}}$ and $\{\beta_1, \ldots, \beta_k\} \subseteq \mathbf{A}^{\mathbf{F}}$ we have that $\delta \subseteq \mathbf{A}$.

Proceeding from Example 6.4, for instance, reconsider the partial assignments $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$. For $\mathbf{A}_2$ we have

$$\mathsf{p}_1(\mathbf{A}_2) = \{\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}, \lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}\} = \{\{\mathbf{T}r, \mathbf{F}\,\mathrm{body}(r_3)\}, \{\mathbf{T}s, \mathbf{F}\,\mathrm{body}(r_3)\}\}.$$

For both nogoods $\delta \in \mathsf{p}_1(\mathbf{A}_2) \subseteq \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ we have $\delta \subseteq \mathbf{A}_2$. On the other hand, for $\mathbf{A}_1$ and $\mathbf{A}_3$, we have $\mathsf{p}_1(\mathbf{A}_1) = \mathsf{p}_1(\mathbf{A}_3) = \emptyset$. Hence, $\mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ are solutions for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, whilst $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is not.

Observe that p_1 is not conflict-optimal. Still, it provides an external propagator that can check whether all atoms in an unfounded set are falsified. ASP

solvers that implements this approach are described in (Lin and Zhao, 2002; Giunchiglia et al., 2006).

Note that, however, p_1 can easily be made conflict-optimal by dropping the requirement of $\mathbf{A}$ being total.

Even though an external propagator for Γ is conflict-optimal, UP on Γ might infer more literals than UP on lazily generated nogoods. To close this gap, we define inference-optimal external propagators. An external propagator p for a set of nogoods Γ is *inference-optimal* if p is conflict-optimal and for any conflict-free assignment $\mathbf{A}$ if $\delta \setminus \mathbf{A} = \{\sigma\}$ such that $\overline{\sigma} \notin \mathbf{A}$ for some $\delta \in \Gamma$ then $\delta' \setminus \mathbf{A} = \{\sigma\}$ for some $\delta' \in \mathsf{p}(\mathbf{A})$. I.e., in addition to the nogoods generated by a conflict-optimal propagator, an inference-optimal one also generates all nogoods from Γ that are unit w.r.t. $\mathbf{A}$.

Example 6.7

Consider the external propagators p_2 and p_3 for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, given through

$$\mathsf{p}_2(\mathbf{A}) = \{\lambda^{p}_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U)} \mid U \in \mathrm{loop}(\Pi_{\mathscr{P}\setminus\mathscr{E}}),\ p \in U,\ \mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) \setminus \mathbf{A}^{\mathbf{F}} = \emptyset\}, \text{ and}$$

$$\mathsf{p}_3(\mathbf{A}) = \mathsf{p}_2(\mathbf{A}) \cup \{\lambda^{p}_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U)} \mid U \in \mathrm{loop}(\Pi_{\mathscr{P}\setminus\mathscr{E}}),\ p \in U \cap \mathbf{A}^{\mathbf{T}},\ |\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) \setminus \mathbf{A}^{\mathbf{F}}| = 1\}.$$

The external propagator p_2 is conflict-optimal because it generates all nogoods from p_1 on any (partial) assignment. In addition, it also generates nogoods of the form $\lambda^{p}_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U)} = \{\mathbf{T}p, \mathbf{F}\beta_1, \ldots, \mathbf{F}\beta_k\}$ where $\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(U) = \{\beta_1, \ldots, \beta_k\}$ and $\{\beta_1, \ldots, \beta_k\} \subseteq \mathbf{A}^{\mathbf{F}}$ but $p \notin \mathbf{A}^{\mathbf{F}}$, i.e., a unit nogood where $\mathbf{F}p$ is unit-resulting.

Proceeding from Example 6.4, for instance, consider the partial assignment $\mathbf{A}'_2 \subset \mathbf{A}_2$ given through

$$\mathbf{A}'_2 = \{\mathbf{F}p, \mathbf{F}\,\mathrm{body}(r_3)\}.$$

Then, we have

$$\mathsf{p}_2(\mathbf{A}'_2) = \{\lambda^{r}_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}, \lambda^{s}_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}\} = \{\{\mathbf{T}r, \mathbf{F}\,\mathrm{body}(r_3)\}, \{\mathbf{T}s, \mathbf{F}\,\mathrm{body}(r_3)\}\},$$

both nogoods are unit, where $\mathbf{F}r$ and $\mathbf{F}s$ are unit-resulting. In fact, p_2 provides an external propagator that can check whether the atoms in an unfounded set have to be falsified, i.e., it encodes FL inference. An ASP solver that implements this approach is described in (Gebser et al., 2007a).

In general, however, p_2 does not generate all nogoods from $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ that are unit w.r.t. an assignment, as much as FL does not infer all consequences from unfounded sets (see Chapter 3). For instance, consider the partial assignment $\mathbf{A}'_1 \subset \mathbf{A}_1$ given through

$$\mathbf{A}'_1 = \{\mathbf{T}r, \mathbf{T}s, \mathbf{T}\mathrm{body}(r_4), \mathbf{T}\mathrm{body}(r_5)\}.$$

Then, we have $\mathsf{p}_2(\mathbf{A}'_1) = \emptyset$. Hence, p_2 is not inference-optimal. On the other hand, we have

$$\mathsf{p}_3(\mathbf{A}'_1) = \{\lambda^r_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}, \lambda^s_{\mathrm{ES}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}(\{r,s\})}\} = \{\{\mathbf{T}r, \mathbf{F}\mathrm{body}(r_3)\}, \{\mathbf{T}s, \mathbf{F}\mathrm{body}(r_3)\}\}.$$

Both nogoods in $\mathsf{p}_3(\mathbf{A}'_1)$ are unit, where $\mathbf{T}\mathrm{body}(r_3)$ is unit-resulting. In fact, p_3 provides an external propagator for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ that encodes FL and BL inference. Moreover, within our example, p_3 is inference-optimal as it generates all unit nogoods from $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$.

The correspondence between external propagation and the set of nogoods it represents is formalised in the following proposition.

Lemma 6.7

Let Δ be a set of nogoods, and p be an external propagator for $\Gamma \subseteq \Delta$. Then, the assignment $\mathbf{A}$ is a solution for Δ if and only if $\mathbf{A}$ is a solution for $\mathsf{p}(\mathbf{A}) \cup (\Delta \setminus \Gamma)$.

Proof. We show both implications of the proposition.

($\Rightarrow$) We prove by contradiction. Suppose $\mathbf{A}$ is a solution for Δ but is conflicting for $\mathsf{p}(\mathbf{A}) \cup (\Delta \setminus \Gamma)$. That means, there is a nogood $\delta \in \mathsf{p}(\mathbf{A}) \cup (\Delta \setminus \Gamma)$ such that $\delta \subseteq \mathbf{A}$. Since $\mathsf{p}(\mathbf{A}) \subseteq \Gamma \subseteq \Delta$, however, we have $\delta \in \Gamma$ and $\mathbf{A}$ cannot be a solution for Δ, contradicting the assumption.

($\Leftarrow$) Again, we prove by contradiction. Suppose $\mathbf{A}$ is a solution for $\mathsf{p}(\mathbf{A}) \cup (\Delta \setminus \Gamma)$ but is conflicting for Δ. That means, there is a nogood $\delta \in \Delta$ such that $\delta \subseteq \mathbf{A}$. Since $\mathbf{A}$ is in particular a solution for $\Delta \setminus \Gamma$, we have $\delta \in \Gamma$. Then, by definition, there is a nogood $\delta' \subseteq \mathbf{A}$ such that $\delta' \in \mathsf{p}(\mathbf{A})$. Hence, $\mathbf{A}$ is conflicting for $\mathsf{p}(\mathbf{A})$. This contradicts our assumption. □

Theorem 6.6 and Lemmata 6.7 and 6.5 yield yet another variant of Lifschitz and Turner's splitting set theorem.

The following proposition states that, given a splitting set $\mathscr{E}$, we can compute the answer sets of a program Π by computing the answer sets of $\Pi_{\mathscr{E}}$ and solutions

for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$. Instead of making all nogoods in $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ explicit, however, an external propagator p can be used to represents a subset thereof.

Theorem 6.8

Let Π be a program over $\mathscr{P}$, $\mathscr{E}$ a splitting set for Π, p an external propagator for $\Gamma \subseteq \Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, $X \subseteq \mathscr{P}$ and $\mathbf{A}$ the assignment given through

$$\begin{aligned}\mathbf{A} = &\{\mathbf{T}p \mid p \in X\} \cup \{\mathbf{F}p \mid p \in \mathscr{P} \setminus X\}\\ &\cup\{\mathbf{T}\mathrm{body}(r) \mid r \in \Pi, \mathrm{body}(r)^+ \subseteq X, \mathrm{body}(r)^- \cap X = \emptyset\}\\ &\cup\{\mathbf{F}\mathrm{body}(r) \mid r \in \Pi, (\mathrm{body}(r)^+ \cap (\mathscr{P} \setminus X))\\ &\qquad\qquad \cup (\mathrm{body}(r)^- \cap X) \neq \emptyset\}.\end{aligned}$$

Then, X is an answer set of Π if and only if

- $X \cap \mathscr{E}$ is an answer set of $\Pi_{\mathscr{E}}$, and
- $\mathbf{A}|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\mathsf{p}(\mathbf{A}) \cup (\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}} \cup \Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}) \setminus \Gamma$.

Proof. The proof follows from Theorem 6.6 and Lemma 6.7. □

Example 6.8

We proceed from Example 6.5 and also reconsider the external p_1 for $\Lambda^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ from Example 6.6. Recall that $\mathbf{A}_1|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\mathbf{A}_3|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ are solutions for $\mathsf{p}_1(\mathbf{A}_1) \cup \Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ and $\mathsf{p}_1(\mathbf{A}_3) \cup \Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$, respectively, whilst $\mathbf{A}_2|_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ is a solution for $\Delta^{\mathscr{E}}_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}$ but not for $\mathsf{p}_1(\mathbf{A}_2)$.

Moreover, recall that for $X_1 = \mathbf{A}_1 \cap \mathscr{P} = \{p, r, s\}$, $X_2 = \mathbf{A}_2 \cap \mathscr{P} = \{q, r, s\}$, and $X_3 = \mathbf{A}_3 \cap \mathscr{P} = \{q\}$ we have that $(X_1 \cap \mathscr{E}) = \{p\}$, and $(X_2 \cap \mathscr{E}) = (X_3 \cap \mathscr{E}) = \{q\}$ are answer sets of $\Pi_{\mathscr{E}}$. Therefore, by Theorem 6.8, only sets X_1 and X_3 are answer sets of Π. Indeed, they are (cf. Example 2.11).

The recursive application of Theorems 6.8, 6.3, and 6.8 allows that the conditions on the answer sets induced by any (sub-) program (over externals) of an ASP encoding can be represented by either unfounded-free solutions to its completion nogoods, or solutions to its completion nogoods and loop nogoods, or solutions to the nogoods generated by an external propagator (plus the remaining nogoods not represented by the external propagator). This provides extreme flexibility, as the programmer can choose between eager or lazy generation of nogoods for any (sub-) program of an ASP encoding.

One of the advantages of LNG over eager encodings is space efficiency. For instance, the worst-case exponential number of loops in a program Π, i.e., nogoods in $\Lambda_\Pi^\emptyset$, makes an explicit representation of Λ_Π infeasible (Lifschitz and Razborov, 2006). On the other hand, as we have argued, FL inference can be implemented in form of a conflict-optimal external propagator that falsifies unfounded sets by generating appropriate nogoods in $\Lambda_\Pi^\emptyset$ on demand (Gebser et al., 2007a) using linear time (Calimeri et al., 2002; Anger et al., 2006). This approach can easily be generalised to any program Π with externals in $\mathscr{E}$, i.e., nogoods in $\Lambda_{\Pi_{\mathscr{P}\setminus\mathscr{E}}}^{\mathscr{E}}$.

In general, the creation of efficient external propagators to represent any given set of nogoods is a challenging task. In the context of CASP, however, we can conveniently draw from CP, a research area that is largely concerned with efficient propagation in solving CSP. In the next sections, we will show how LNG can be applied to CASP, in particular, how encodings for variables' domains and global constraints can be captured by external propagators. Following the idea of Ohrimenko et al. (2009) we will apply CP techniques to propagate constraints and then generating lazy nogoods that represent the underlying inference.

Encoding Variable's Domain via External Propagation

We start with external propagators for the nogoods represented by ASP-VALUE[V] from Section 4.2. Recall, ASP-VALUE[V] introduces atoms of the form $[\![\mathsf{v} = i]\!]$ to indicate whether $\mathsf{v} = i$ is possible, where $\mathsf{v} \in \mathsf{V}$ is a variable and $i \in \mathrm{dom}(\mathsf{v})$ a value from its domain. The conditions induced by ASP-VALUE[V] ensure any conflict-free assignment $\mathbf{A}$ represents a consistent set of domains for the values in V.

For reasons of space complexity we have previously transformed the cardinality constraint rules from ASP-VALUE[V] into a set of rules following the idea of Sinz (2005), i.e., encoding sequential counters. With LNG, however, we can employ an external propagator for the nogoods represented by the quadratic size decomposition into integrity constraints (cf. Section 2.5). For instance, Algorithm 6.1 provides a specification of an inference-optimal external propagator for this task. It takes an assignment $\mathbf{A}$ and returns a set of lazily generated nogoods, initialised in Line 1, that are unit or conflicting. Lines 2–3 ensure that if v is assigned a value i then all other values are removed from its domain, while Lines 4–5 deal with the condition that there is at least one value that can be assigned to v. This procedure can be made very efficient, for instance, by using *watched literals* (Moskewicz et al., 2001; Gent, 2013).

Input : A Boolean assignment $\mathbf{A}$.
Output: A set of lazily generated nogoods.

1 $\nabla \leftarrow \emptyset$ *// set of lazily generated nogoods*
2 **if** $[[\mathsf{v} = i]] \in \mathbf{A^T}$ for some $i \in \mathrm{dom}(\mathsf{v})$ **then**
3 $\quad \nabla \leftarrow \{\{\mathbf{T}[[\mathsf{v} = i]], \mathbf{T}[[\mathsf{v} = j]]\} \mid j \in \mathrm{dom}(\mathsf{v}) \setminus \{i\},\ [[\mathsf{v} = j]] \notin \mathbf{A^F}\}$
4 **if** $[[\mathsf{v} = i]] \notin \mathbf{A^T}$ for some $i \in \mathrm{dom}(\mathsf{v}) \wedge \forall j \in \mathrm{dom}(\mathsf{v}) \setminus \{i\}\ [[\mathsf{v} = j]] \in \mathbf{A^F}$ **then**
5 $\quad \nabla \leftarrow \{\{\mathbf{F}[[\mathsf{v} = j]] \mid j \in \mathrm{dom}(\mathsf{v})\}\}$
6 **return** ∇

Algorithm 6.1: An external propagator for nogoods represented by ASP-VALUE[V].

Recall that other options for representing the variables' domains, for instance, the bound encoding ASP-BOUND[V], the range encoding ASP-RANGE[V], or variants there-of are also possible. As with ASP-VALUE[V] we can employ external propagators for $\Delta^{\emptyset}_{\text{ASP-BOUND}[\mathsf{V}]}$ and $\Delta^{\emptyset}_{\text{ASP-RANGE}[\mathsf{V}]}$, respectively. In a similar fashion, this carries over to mixed encodings.

Encoding Constraints via External Propagation

As described in Chapter 4, we can view atoms from the above encodings as primitive constraints because all constraints can be decomposed into a set of rules over them, by using one of our generic encodings (Sections 4.2–4.5), or specialised encodings that simulate the inference of constraint propagation via ASP inference (Sections 5.1–5.3). Either way, to maintain a certain level of local consistency, the changes to the variables' domains are described in terms of primitive constraints. Based on this insight, we can create an external propagator for any constraint by simply modifying an existing propagation algorithm for the constraint to make its inferences explicit in form of nogoods over primitive constraints.

Algorithm 6.2 specifies an external propagator for the nogoods represented by an ASP encoding of the ALL-DIFFERENT constraint $c = \text{ALL-DIFFERENT}(\{\mathsf{v}_1, \ldots, \mathsf{v}_n\})$. Provided with an assignment $\mathbf{A}$, it starts with an empty set of lazily generated nogoods, followed by a distinction into two cases. First, if the constraint must not be violated, i.e., $[[c]] \in \mathbf{A^T}$, then for each variable in the scope of the constraint that has a value assigned, a nogood is generated that asserts the removal of this value from the domain of all other variables in the scope of the constraint (Lines 3–4). On the other hand, if the constraint is set to be violated or the constraint atom is unassigned, i.e., $[[c]] \notin \mathbf{A^T}$, the algorithm checks whether two variables in the scope

Input : A Boolean assignment $\mathbf{A}$.
Output: A set of lazily generated nogoods.

1 $\nabla \leftarrow \emptyset$ *// set of lazily generated nogoods*
2 **if** $[\![c]\!] \in \mathbf{A^T}$ **then**
3 **foreach** $\mathsf{v} \in \mathrm{scope}(c)$ s.t. $[\![\mathsf{v} = i]\!] \in \mathbf{A^T}$ for some $i \in \mathrm{dom}(\mathsf{v})$ **do**
4 $\nabla \leftarrow \nabla \cup \{\{\mathbf{T}[\![c]\!], \mathbf{T}[\![\mathsf{v} = i]\!], \mathbf{T}[\![\mathsf{w} = i]\!]\} \mid \mathsf{w} \in \mathrm{scope}(c) \setminus \{\mathsf{v}\}$
$i \in \mathrm{dom}(\mathsf{w}),\ [\![\mathsf{w} = i]\!] \notin \mathbf{A^F}\}$
5 **else**
6 **foreach** $\mathsf{v} \in \mathrm{scope}(c)$ s.t. $[\![\mathsf{v} = i]\!] \in \mathbf{A^T}$ for some $i \in \mathrm{dom}(\mathsf{v})$ **do**
7 **if** $\mathsf{w} \in \mathrm{scope}(c) \setminus \{\mathsf{v}\}$ s.t. $[\![\mathsf{w} = i]\!] \in \mathbf{A^T}$ **then**
8 **if** $[\![c]\!] \notin \mathbf{A^F}$ **then**
9 $\nabla \leftarrow \{\{\mathbf{T}[\![c]\!], \mathbf{T}[\![\mathsf{v} = i]\!], \mathbf{T}[\![\mathsf{w} = i]\!]\}\}$
10 **return** ∇
11 **if** $\forall \mathsf{v} \in \mathrm{scope}(c)\ \exists i \in \mathrm{dom}(\mathsf{v})$ s.t. $[\![\mathsf{v} = i]\!] \in \mathbf{A^T}$ **then**
12 $\nabla \leftarrow \{\{\mathbf{F}[\![c]\!]\} \cup \{\mathbf{T}[\![\mathsf{v} = i]\!] \mid \mathsf{v} \in \mathrm{scope}(c), i \in \mathrm{dom}(\mathsf{v}),\ [\![\mathsf{v} = i]\!] \in \mathbf{A^T}\}\}$
13 **return** ∇

Algorithm 6.2: An external propagator for nogoods represented by ASP-ALL-DIFFERENT$[\{\mathsf{v}_1, \ldots, \mathsf{v}_n\}]$ encoding $c = \text{ALL-DIFFERENT}(\{\mathsf{v}_1, \ldots, \mathsf{v}_n\})$.

of the constraint have the same value assigned (Lines 6–10). If this is the case, the ALL-DIFFERENT constraint is violated and a nogood asserting that the constraint atom is set to false will be returned (unless $[\![c]\!] \in \mathbf{A^T}$, in which case the constraint atom is already false). If, however, no such pair of variables can be found and all variables in the scope of the constraint have a value assigned, then the ALL-DIFFERENT condition is satisfied and a nogood is generated that asserts the truth of the corresponding constraint atom (Lines 11–12).

Example 6.9

Reconsider the constraint $c = \text{ALL-DIFFERENT}(\{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4\})$ from Example 5.1 and the assignment $\mathbf{A}_1$ given through

$$\mathbf{A}_1 = \left\{ \begin{array}{llll} \mathbf{F}[\![\mathsf{v}_1 = 1]\!], & \mathbf{T}[\![\mathsf{v}_1 = 2]\!], & \mathbf{F}[\![\mathsf{v}_1 = 3]\!], & \mathbf{F}[\![\mathsf{v}_1 = 4]\!], \\ & & & \mathbf{F}[\![\mathsf{v}_2 = 4]\!], \\ \mathbf{F}[\![\mathsf{v}_4 = 1]\!], & & & \mathbf{F}[\![\mathsf{v}_4 = 4]\!], \\ \mathbf{T}[\![c]\!] & & & \end{array} \right\}$$

Verify that **A** represents the following domains:

	1	2	3	4
v_1		✓		
v_2	✓	✓	✓	
v_3	✓	✓	✓	✓
v_4		✓	✓	

Let p_c be the external propagator specified in Algorithm 6.2 for the nogoods represented by ASP-ALL-DIFFERENT$[\{v_1,\ldots,v_4\}]$. The application of p_c yields the following nogoods:

$$p_c(\mathbf{A}_1) = \nabla_1 = \left\{ \begin{array}{l} \{\mathbf{T}[[c]], \mathbf{T}[[v_1 = 2]], \mathbf{T}[[v_2 = 2]]\}, \\ \{\mathbf{T}[[c]], \mathbf{T}[[v_1 = 2]], \mathbf{T}[[v_3 = 2]]\}, \\ \{\mathbf{T}[[c]], \mathbf{T}[[v_1 = 2]], \mathbf{T}[[v_4 = 2]]\} \end{array} \right\}$$

This encodes the removal of the value 2 from the domains of v_2, v_3, and v_4 on the grounds that the constraint c is not to be violated and v_1 is already assigned 2. In turn, UP updates the domains. That is,

$$\mathrm{UP}^*(\Delta_{\text{ASP-VALUE}[\{v_1,\ldots,v_4\}]} \cup \nabla_1, \mathbf{A}_1) = \mathbf{A}_2 = \mathbf{A}_1 \cup \left\{ \begin{array}{l} \mathbf{F}[[v_2 = 2]], \mathbf{F}[[v_3 = 2]], \\ \mathbf{F}[[v_4 = 2]] \end{array} \right\}.$$

Let p_V be the external propagator specified in Algorithm 6.1 for the nogoods represented by ASP-VALUE$[\{v_1,\ldots,v_4\}]$. The application of p_V yields the following nogoods:

$$p_V(\mathbf{A}_2) = \nabla_2 = \{\{\mathbf{F}[[v_4 = 1]], \mathbf{F}[[v_4 = 2]], \mathbf{F}[[v_4 = 3]], \mathbf{F}[[v_4 = 4]]\}\}$$

This encodes the assignment of the only remaining value 3 in the domains of v_4 on the grounds that at least one value must be in the domain of v_1. As before, UP performs the inference encoded. That is,

$$\mathrm{UP}^*(\Delta_{\text{ASP-VALUE}[\{v_1,\ldots,v_4\}]} \cup \nabla_2, \mathbf{A}_2) = \mathbf{A}_3 = \mathbf{A}_2 \cup \{\mathbf{T}[[v_4 = 3]]\}.$$

Further propagation is possible

$$p_c(\mathbf{A}_3) = \nabla_3 = \left\{ \begin{array}{l} \{\mathbf{T}[[c]], \mathbf{T}[[v_4 = 3]], \mathbf{T}[[v_2 = 3]]\}, \\ \{\mathbf{T}[[c]], \mathbf{T}[[v_4 = 3]], \mathbf{T}[[v_3 = 3]]\} \end{array} \right\}$$

where p_c encodes the removal of 3 from the domains of v_2 and v_3, again, performed by UP. Verify that

$$\mathrm{UP}^*(\Delta_{\text{ASP-VALUE}[\{\mathsf{v}_1,\ldots,\mathsf{v}_4\}]} \cup \nabla_3, \mathbf{A}_3) = \mathbf{A}_4 = \mathbf{A}_3 \cup \{\mathbf{F}[[\mathsf{v}_2 = 3]], \mathbf{F}[[\mathsf{v}_3 = 3]]\}$$

and then

$$\mathsf{p}_\mathsf{V}(\mathbf{A}_4) = \nabla_4 = \{\{\mathbf{F}[[\mathsf{v}_2 = 1]], \mathbf{F}[[\mathsf{v}_2 = 2]], \mathbf{F}[[\mathsf{v}_2 = 3]], \mathbf{F}[[\mathsf{v}_2 = 4]]\}\}$$

i.e., p_V encodes that v_2 must be assigned the remaining value in its domain. In turn,

$$\mathrm{UP}^*(\Delta_{\text{ASP-VALUE}[\{\mathsf{v}_1,\ldots,\mathsf{v}_4\}]} \cup \nabla_4, \mathbf{A}_4) = \mathbf{A}_5 = \mathbf{A}_4 \cup \{\mathbf{T}[[\mathsf{v}_2 = 1]]\}$$

UP assign the value 1 to v_2. We can repeat this process, until a fixpoint is reached. To proceed,

$$\mathsf{p}_c(\mathbf{A}_5) = \nabla_5 = \left\{ \ \{\mathbf{T}[[c]], \mathbf{T}[[\mathsf{v}_2 = 1]], \mathbf{T}[[\mathsf{v}_3 = 1]]\} \ \right\}$$

and then

$$\mathrm{UP}^*(\Delta_{\text{ASP-VALUE}[\{\mathsf{v}_1,\ldots,\mathsf{v}_4\}]} \cup \nabla_5, \mathbf{A}_5) = \mathbf{A}_6 = \mathbf{A}_5 \cup \{\mathbf{F}[[\mathsf{v}_3 = 1]]\}$$

i.e., the value 1 is removed from v_3, and then

$$\mathsf{p}_\mathsf{V}(\mathbf{A}_6) = \nabla_6 = \{\{\mathbf{F}[[\mathsf{v}_4 = 1]], \mathbf{F}[[\mathsf{v}_4 = 2]], \mathbf{F}[[\mathsf{v}_4 = 3]], \mathbf{F}[[\mathsf{v}_4 = 4]]\}\}$$

and

$$\mathrm{UP}^*(\Delta_{\text{ASP-VALUE}[\{\mathsf{v}_1,\ldots,\mathsf{v}_4\}]} \cup \nabla_6, \mathbf{A}_6) = \mathbf{A}_7 = \mathbf{A}_6 \cup \{\mathbf{T}[[\mathsf{v}_3 = 4]]\}$$

i.e., the value 4 is assigned to v_3. Finally, verify that $\mathsf{p}_c(\mathbf{A}_7) = \mathsf{p}_\mathsf{V}(\mathbf{A}_7) = \emptyset$, i.e., no further propagation is possible. This leaves the assignment $\mathbf{A}_7$, where

$$\mathbf{A}_7 = \left\{ \begin{array}{llll} \mathbf{F}[[\mathsf{v}_1 = 1]], & \mathbf{T}[[\mathsf{v}_1 = 2]], & \mathbf{F}[[\mathsf{v}_1 = 3]], & \mathbf{F}[[\mathsf{v}_1 = 4]], \\ \mathbf{T}[[\mathsf{v}_2 = 1]], & \mathbf{F}[[\mathsf{v}_2 = 2]], & \mathbf{F}[[\mathsf{v}_2 = 3]], & \mathbf{F}[[\mathsf{v}_2 = 4]], \\ \mathbf{F}[[\mathsf{v}_3 = 1]], & \mathbf{F}[[\mathsf{v}_3 = 2]], & \mathbf{F}[[\mathsf{v}_3 = 3]], & \mathbf{T}[[\mathsf{v}_3 = 4]], \\ \mathbf{F}[[\mathsf{v}_4 = 1]], & \mathbf{F}[[\mathsf{v}_4 = 2]], & \mathbf{T}[[\mathsf{v}_4 = 3]], & \mathbf{F}[[\mathsf{v}_4 = 4]]. \\ \mathbf{T}[[c]] & & & \end{array} \right\}$$

representing the following set of arc consistent domains:

	1	2	3	4
v_1		✓		
v_2	✓			
v_3				✓
v_4			✓	

Notice that Algorithm 6.2 generates nogoods that are either unit or conflicting w.r.t. **A**. This is best practice as it will enable UP to perform the encoded inference within its next iteration, thus pruning (the representation of) some variable's domain, or, for instance, the truth value of the constraint atom representing a constraint. In this manner, constraint propagation via external propagation, and ASP inference can be repeated until a fixpoint is reached or a conflict is encountered. By generating a conflicting nogood, for instance, an external propagator can yield that the underlying constraint is violated (with respect to the assignment to the associated constraint atom). If $[\![c]\!] \in \mathbf{A}^{\mathbf{T}}$, observe that the external propagator specified in Algorithm 6.2 enforces arc consistency on the binary decomposition of the original constraint. Other propagators are also possible. In fact, external propagation provides a form of hybridisation that allows programmers to select encodings which propagate better, but were previously avoided for space-related reasons. For instance, in Section 5.1 we have described ASP encodings of the ALL-DIFFERENT constraint such that ASP inference maintains bound or range consistency. As discussed in the previous chapter, a constraint propagator that can achieve domain consistency exists (Régin, 1994) but it cannot be simulated efficiently with UP using eager encodings (Bessière et al., 2009b). Because of the fact that external propagators generate nogoods only on demand, however, we can implicitly represent encodings via LNG that are otherwise impractical. We will exercise this option in Section 6.4.

6.3 Decision Algorithm

To facilitate conflict-driven constraint answer set solving via LNG, we here develop a decision procedure for answer set solving with LNG based on CDNL-ASP (Gebser et al., 2007a). We will inherit all features of CDNL, including CONFLICTANALYSIS according to the *First-UIP* scheme (Zhang et al., 2001). This will guide dynamic search heuristics and backjumping by iteratively resolving a conflicting nogood against other nogoods until a *unique implication point* is obtained. Recording the

resulting nogood enables conflict-driven learning, which can further prune the search space. Similar to CDNL, for controlling the set of recorded nogoods, deletion strategies can be applied (cf. Moskewicz et al., 2001). In contrast to CDNL-ASP we integrate external propagators that perform LNG in order to represent conditions on the answer sets of a program that are not encoded eagerly. By making them explicit (in particular, when they propagate new information), lazily generated nogoods can contribute to CONFLICTANALYSIS like their eager counterpart, i.e., the nogoods involved in this process can stem from eagerly and lazily generated encodings. This can improve propagation and contribute to lookback-based search heuristics. Different to eagerly encoded nogoods, however, the amount of lazily generated nogoods can be controlled via deletion.

Conflict-driven Nogood Learning with Lazy Nogood Generation

Algorithm 6.3 specifies our main procedure, CDNL-LNG. It takes a constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ and a setting $(\psi, \pi, \mathit{mode})$, where

- ψ is a function that maps each variable $\mathsf{v} \in \mathsf{V}$ to an ASP encoding Ψ_v of its domain, and each constraint atom $c \in \mathrm{atom}(\mathsf{C})$ to an ASP encoding Ψ_c of the corresponding constraint $\mathrm{constraint}(c)$,

- π is a function that maps each variable $\mathsf{v} \in \mathsf{V}$ to an external propagator p_v for the nogoods represented by Ψ_v, and each constraint atom $c \in \mathrm{atom}(\mathsf{C})$ an external propagator p_c for the nogoods represented by Ψ_c, and

- *mode* is a function that maps each variable $\mathsf{v} \in \mathsf{V}$ and each constraint atom $c \in \mathrm{atom}(\mathsf{C})$ to the values *eager* or *lazy*. The mapping determines whether the nogoods represented by Ψ_v and Ψ_c are encoded eagerly or lazily, respectively.

Given this, Algorithm 6.3 initialises the ASP encoding Ψ of all components of the corresponding CSP that are configured to be eagerly encoded in Line 1, based on our theoretical foundations laid in Chapter 4. The externals $\mathscr{E}$ of $\Pi \cup \Psi$ are computed in Line 2, that are, atoms in the ASP encoding of the remaining components of the CSP. These components are the ones configured to be lazily encoded via external propagators. The algorithm collects external propagators that stem from variables in P_V (Line 3) and constraints in P_C (Line 4). Then, it proceeds with an empty assignment $\mathbf{A}$ and an empty set ∇ that will store recorded nogoods, includ-

Input : A constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ and a setting $(\psi, \pi, \mathit{mode})$.
Output: A constraint answer set of $\mathbb{P}$ if one exists.

1 $\Psi \leftarrow \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{eager}} \psi(\mathsf{v}) \cup \bigcup_{c\in\mathrm{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{eager}} \psi(c)$
2 $\mathscr{E} \leftarrow \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \mathrm{atom}(\psi(\mathsf{v})) \cup \bigcup_{c\in\mathrm{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{lazy}} \mathrm{atom}(\psi(c))$
3 $\mathsf{P}_\mathsf{V} \leftarrow \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \pi(\mathsf{v})$
4 $\mathsf{P}_\mathsf{C} \leftarrow \bigcup_{c\in\mathrm{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{lazy}} \pi(c)$
5 $\mathbf{A} \leftarrow \emptyset$
6 $\nabla \leftarrow \emptyset$
7 $dl \leftarrow 0$
8 **loop**
9 $\quad(\mathbf{A}, \nabla) \leftarrow$ PROPAGATION$(\Pi \cup \Psi, \mathsf{P}_\mathsf{V}, \mathsf{P}_\mathsf{C}, \nabla, \mathbf{A})$
10 $\quad$**if** $\delta \subseteq \mathbf{A}$ for some $\delta \in \Delta^{\mathscr{E}}_{\Pi\cup\Psi} \cup \nabla$ **then**
11 $\quad\quad$**if** $dl = 0$ **then return** no constraint answer set
12 $\quad\quad(\varepsilon, k) \leftarrow$ CONFLICTANALYSIS$(\delta, \Pi \cup \Psi, \nabla, \mathbf{A})$
13 $\quad\quad\nabla \leftarrow \nabla \cup \{\varepsilon\}$
14 $\quad\quad\mathbf{A} \leftarrow \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dl(\sigma)\}$
15 $\quad\quad dl \leftarrow k$
16 $\quad$**else if** $\mathbf{A^T} \cup \mathbf{A^F} = \mathrm{atom}(\Pi \cup \Psi) \cup \mathrm{body}(\Pi \cup \Psi) \cup \mathscr{E}$ **then**
17 $\quad\quad$**return** $(\mathbf{A^T} \cap \mathrm{atom}(\Pi), \mathrm{assign}_{\mathsf{V},\mathsf{D}}(\mathbf{A^T} \cap \bigcup_{\mathsf{v}\in\mathsf{V}} \mathrm{atom}(\psi(\mathsf{v}))))$
18 $\quad$**else**
19 $\quad\quad\sigma_d \leftarrow$ SELECT$(\Pi \cup \Psi, \mathscr{E}, \nabla, \mathbf{A})$
20 $\quad\quad\mathbf{A} \leftarrow \mathbf{A} \cup \{\sigma_d\}$
21 $\quad\quad dl \leftarrow dl + 1$

Algorithm 6.3: CDNL-LNG

ing lazily generated nogoods from external propagators. The *decision level dl* is initialised with 0.

The following loop is very similar to Gebser et al.'s description of CDNL-ASP, adjusted to our setting. First, PROPAGATION (Line 9) extends $\mathbf{A}$ and ∇, as described in the next section. If this encounters a conflict (Line 10), where $\Delta^{\mathscr{E}}_{\Pi\cup\Psi} = \Delta^{\mathrm{atom}(\mathsf{C})}_{\Pi} \cup \Delta^{\mathscr{E}}_{\Psi}$ by Lemma 6.1, then the CONFLICTANALYSIS procedure generates a conflicting nogood ε by exploiting interdependencies between the nogoods represented by $\Pi \cup \Psi$ and ∇ through CONFLICTANALYSIS, and determines a decision level k at which to continue search. Then, ε is added to the set of recorded nogoods ∇ in Line 12. This can prune the search space and lead to more propagation. Lines 13–15 account for backjumping to level k. Thereafter ε is unit and triggers inference in the next round of propagation. If CONFLICTANALYSIS, however, yields a conflict at level 0, no answer set exists (Line 11). Furthermore, we distinguish the cases of

Input : A program Π with externals $\mathscr{E}$, two sets of external propagators P and Q, recorded nogoods ∇, and an assignment $\mathbf{A}$.
Output: An extended assignment and set of recorded nogoods.

```
1  loop
2      A ← UP*(Δ_Π^ℰ ∪ ∇, A)
3      if δ ⊆ A for some δ ∈ Δ_Π^ℰ ∪ ∇ then return (A, ∇)
4      foreach p ∈ P do
5          Γ ← p(A)                          // high priority external propagation
6          if Γ ≠ ∅ then break
7      if Γ = ∅ then foreach p ∈ Q do
8          Γ ← p(A)                          // low priority external propagation
9          if Γ ≠ ∅ then break
10     if Γ = ∅ then
11         Γ ← UnfoundedSetPropagation(Π, A)
12     if Γ = ∅ then return (A, ∇)
13     ∇ ← ∇ ∪ Γ
```

Algorithm 6.4: PROPAGATION

a total assignment (Lines 16–17) and a partial one (Lines 18–21). In case of a total assignment, the atoms in $\mathbf{A}^{\mathbf{T}}$ represent a constraint answer set of Ψ, where $\mathbf{A}^{\mathbf{T}} \cap \mathrm{atom}(\Pi)$ is an answer set of $\Pi(\mathrm{sat}_{\mathsf{C}}(A))$ and $A = \mathrm{assign}_{\mathsf{V},\mathsf{D}}(\mathbf{A}^{\mathbf{T}} \cap \bigcup_{\mathsf{v}\in\mathsf{V}} \mathrm{atom}(\psi(\mathsf{v})))$ is an assignment to the variables in V such that $(A, \mathrm{sat}_{\mathsf{C}}(A))$ is a configuration of the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$. In the other case, $\mathbf{A}$ is partial and no nogood is conflicting. Then, a decision literal σ_d is selected by some heuristic, added to $\mathbf{A}$, and the decision level is incremented.

Whilst the CONFLICTANALYSIS and SELECT procedures are similar to the ones in CDNL-ASP, we extend PROPAGATION to accommodate LNG.

Nogood Propagation with Lazy Nogood Generation

A specification of our PROPAGATION procedure is shown in Algorithm 6.4. It works on a program Π with externals $\mathscr{E}$, two sets of external propagators P and Q, a set of recorded nogoods ∇, and an assignment $\mathbf{A}$. PROPAGATION interleaves UP on nogoods $\Delta_{\Pi}^{\mathscr{E}}$ and recorded nogoods ∇. The latter may include lazily generated nogoods from external propagators.

We start with UP (Line 2), resulting either in a conflict, i.e., some nogood is conflicting (Line 3) w.r.t. $\mathbf{A}$, or in a fixpoint that extends $\mathbf{A}$ with unit-resulting liter-

als. If there is no conflict, PROPAGATION performs external propagation following some priority (Lines 4–9), where precedence is given to external propagators in P over external propagators in Q. Typically, the ones in P represent the variables' domains, and the ones in Q represent constraints. (We want to update domains and, in particular, discover an empty domain before propagating constraints.) In principle, additional priority queues or an elaborate scheduler are also possible. Anyway, based on $\mathbf{A}$, each external propagator may encode inference in a set of lazily generated nogoods Γ which is added to the set of recorded nogoods ∇ at the end of the loop in Line 13. The UNFOUNDEDSETPROPAGATION procedure (Line 11; cf. Gebser et al., 2007a) works similarly, and ensures that $\mathbf{A}^{\mathbf{F}}$ does not contain atoms from any unfounded set, for instance, using an implementation of FL or WFN. In addition, the procedure may employ our approximations for WFJ or WFD, as discribed in Chapter 3.

It is important to note that external propagation and UNFOUNDEDSETPROPAGATION is interleaved by UP. This is to assign unit-resulting literals immediately and detect conflicts early. Our algorithm also favours external propagation over UNFOUNDEDSETPROPAGATION. Although the letter may have a better asymptotic time complexity than some external propagators in practical applications, our design decision is grounded on the observation that external propagators can falsify external support, i.e., UNFOUNDEDSETPROPAGATION may depend on the results from external propagation.

Soundness and Completeness

We now establish soundness and completeness of CDNL-LNG. In fact, since it is a decision algorithm, completeness is due to termination. Soundness and termination of CDNL-LNG largely follow from the fact that CDNL-ASP is sound and terminates (Gebser, 2011). The relationship between CDNL-LNG and CDNL-ASP is particularly obvious in the case when all CP constructs are eagerly encoded.

Definition 6.5: Eager Setting

For a constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$, the setting $(\psi, \pi, \mathit{mode})$ such that $\mathit{mode}(\mathsf{c}) = \mathit{eager}$ for each $\mathsf{v} \in \mathsf{V}$ and $\mathit{mode}(c) = \mathit{eager}$ for each $c \in \mathrm{atom}(\mathsf{C})$ is called the *eager setting*.

The following proposition establishes soundness and termination of CDNL-LNG under eager settings.

Theorem 6.9

Let $\mathbb{P}$ be a constraint program and $(\psi, \pi, \mathit{mode})$ the eager setting. Then CDNL-LNG terminates, and returns a constraint answer set of $\mathbb{P}$ if and only if some constraint answer set of $\mathbb{P}$ exists.

Proof. Let $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$. By recursive application of Lifschitz and Turner's splitting set theorem, the program Ψ constructed in Line 1 of CDNL-LNG is an ASP encoding of the CSP $(\mathsf{V}, \mathsf{D}, \mathsf{C})$. Furthermore, we have $\mathscr{E} = \emptyset$ (Line 2), $\mathsf{P}_\mathsf{V} = \emptyset$ (Line 3), $\mathsf{P}_\mathsf{C} = \emptyset$ (Line 4), and the equation $\Delta^{\emptyset}_{\Pi\cup\Psi} = \Delta_{\Pi\cup\Psi}$ (Line 10). Given this, CDNL-LNG works just like CDNL-ASP, i.e., our added routines do not operate. By Theorem 4.1, the answer sets of $\Pi \cup \Psi$ coincide with the constraint answer sets of $\mathbb{P}$. Then, the soundness and termination of CDNL-LNG directly follows from the soundness and termination of CDNL-ASP Gebser (2011). □

We now turn our attention to any setting different from the eager setting, i.e., settings $(\psi, \pi, \mathit{mode})$ with $\mathit{mode}(\mathsf{v}) = \mathit{eager}$ for some $\mathsf{v} \in \mathsf{V}$ or $\mathit{mode}(c) = \mathit{lazy}$ for some $c \in \mathrm{atom}(\mathsf{C})$. Soundness with respect to the decision problem of the existence of a constraint answer set is obtained from PROPAGATION and CONFLICTANALYSIS exploiting the available information, without drawing incorrect conclusions. As CONFLICTANALYSIS remains unchanged from CDNL, respectively CDNL-ASP, we proceed with analysing our LNG extensions to PROPAGATION.

Before we begin, we make one reasonable assumption to avoid multiple case distinctions in the following proofs. The assumption is that and that UNFOUNDEDSETPROPAGATION returns at least those loop nogoods of a given ASP encoding which are generated by the external propagator for the nogoods represented the model.

Assumption 6.6

Let Π be a program with externals $\mathscr{F}$, $\mathscr{E}$ be a splitting set for Π, p be an external propagator for $\Delta^{\mathscr{F}}_{\Pi_\mathscr{E}} \cup \Lambda^{\mathscr{F}}_{\Pi_\mathscr{E}}$, and $\mathbf{A}$ be an assignment. We assume that $\textsc{UnfoundedSetPropagation}(\Pi, \mathbf{A}) \subseteq \mathsf{p}(\mathbf{A})$.

In practice, UNFOUNDEDSETPROPAGATION is not required to satisfy the above requirement for arbitrary assignments, but eventually produces some conflicting loop nogood every total assignment that is not a solution for the set of loop nogoods. We have provided examples for conflict-optimal and inference-optimal unfounded set inference in Section 6.2.

The following proposition establishes that the assignment and the set of recorded nogoods returned by PROPAGATION is bounded by the elements returned by PROPAGATION under the eager setting. In other words, PROPAGATION with LNG always exploits less information than PROPAGATION under the corresponding eager encoding.

Lemma 6.10

Let $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ be a constraint program and $(\psi, \pi, \mathit{mode})$ be a setting. Let

$$\Psi_{\mathsf{V}} = \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{eager}} \psi(\mathsf{v}) \text{ and } \hat{\Psi}_{\mathsf{V}} = \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \psi(\mathsf{v})$$

be the eager and lazy encodings of the variables' domains, and let

$$\Psi_{\mathsf{C}} = \bigcup_{c\in\mathrm{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{eager}} \psi(c) \text{ and } \hat{\Psi}_{\mathsf{C}} = \bigcup_{c\in\mathrm{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{lazy}} \psi(c)$$

be the eager and lazy encodings of the constraints, such that $\Psi = \Psi_{\mathsf{V}} \cup \hat{\Psi}_{\mathsf{V}} \cup \Psi_{\mathsf{C}} \cup \hat{\Psi}_{\mathsf{C}}$ as constructed in Line 1 of CDNL-LNG under the eager setting. Let P_{V} and P_{C} be the set of external propagators constructed in Lines 3 and 4 under the setting $(\psi, \pi, \mathit{mode})$.

Let ∇ be a set of nogoods, $\mathbf{A}$ be an assignment, and $\mathbf{A}_e$ be the assignment returned in Line 12 of PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$. Then, the execution of PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A})$ returns in Line 12 with an assignment $\mathbf{A}_l$ such that $\mathbf{A}_l \subseteq \mathbf{A}_e$.

Proof. Before we begin, let $\mathscr{E} \subseteq \mathrm{atom}(\Psi_{\mathsf{V}} \cup \hat{\Psi}_{\mathsf{V}})$ be the externals of Ψ_c and let $\hat{\mathscr{E}} \subseteq \mathrm{atom}(\Psi_{\mathsf{V}} \cup \hat{\Psi}_{\mathsf{V}})$ be the externals of $\hat{\Psi}_c$, where $c \in \mathrm{atom}(\mathsf{C})$. Then, by construction, we have that P_{V} are the propagators that represent the set of nogoods $\Delta_{\hat{\Psi}_{\mathsf{V}}} \cup \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \Lambda_{\hat{\Psi}_{\mathsf{v}}}$, and P_{C} are the propagators that represent the set of nogoods $\Delta^{\hat{\mathscr{E}}}_{\hat{\Psi}_{\mathsf{C}}} \cup \bigcup_{c\in\mathrm{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{lazy}} \Lambda^{\hat{\mathscr{E}}}_{\hat{\Psi}_c}$.

Now, let PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$ return the pair $(\mathbf{A}_e, \nabla_e)$ in Line 12. We show that the execution of PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A})$ returns a pair $(\mathbf{A}_l, \nabla_l)$ in Line 12 such that $\mathbf{A}_l \subseteq \mathbf{A}_e$ by proof of contradiction.

Suppose, PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A})$ returns $(\mathbf{A}_l, \nabla_l)$ such that $\mathbf{A}_l \setminus \mathbf{A}_e \neq \emptyset$. Then, $\mathbf{A}_l$ extends $\mathbf{A}_e \cap \mathbf{A}_l$ by a literal σ that is unit-resulting w.r.t. $\mathbf{A}_e \cap \mathbf{A}_l$ in Line 2. In particular, the literal σ belongs to some nogood δ from the set of completion nogoods of $\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}$ or recorded nogoods $\Delta^{(\mathrm{atom}(\mathsf{C})\setminus\mathrm{atom}(\Psi_{\mathsf{C}}))\cup(\mathscr{E}\setminus\mathrm{atom}(\Psi_{\mathsf{V}}))}_{\Pi\cup\Psi_{\mathsf{V}}\cup\Psi_{\mathsf{C}}} \cup \nabla_l$ that is unit with respect to $\mathbf{A}_e \cap \mathbf{A}_l$. By Lemma 6.1, we have that the equation

$\Delta^{(\mathsf{atom}(\mathsf{C})\setminus\mathsf{atom}(\Psi_\mathsf{C}))\cup(\mathscr{E}\setminus\mathsf{atom}(\Psi_\mathsf{V}))}_{\Pi\cup\Psi_\mathsf{V}\cup\Psi_\mathsf{C}} \cup \nabla_l = \Delta^{\mathsf{atom}(\mathsf{C})}_{\Pi} \cup \Delta^{\mathscr{E}}_{\Psi_\mathsf{C}} \cup \Delta_{\Psi_\mathsf{V}} \cup \nabla_l$ holds. Next, observe that the set of recorded nogoods ∇_l augments ∇ via Line 13 by

- loop nogoods from $\Lambda^{(\mathsf{atom}(\mathsf{C})\setminus\mathsf{atom}(\Psi_\mathsf{C}))\cup(\mathscr{E}\setminus\mathsf{atom}(\Psi_\mathsf{V}))}_{\Pi\cup\Psi_\mathsf{V}\cup\Psi_\mathsf{C}}$ (Line 11), or
- lazily generated nogoods $\mathsf{p}(\mathbf{A}')$ for some external propagator $\mathsf{p} \in \mathsf{P}_\mathsf{V} \cup \mathsf{P}_\mathsf{C}$, where $\mathbf{A}'$ is the assignment in some iteration of the loop, $\mathbf{A} \subseteq \mathbf{A}' \subseteq (\mathbf{A}_e \cap \mathbf{A}_l)$.

Since $\mathsf{p}(\mathbf{A}') \subseteq \Delta^{\hat{\mathscr{E}}}_{\hat{\Psi}_\mathsf{C}} \cup \bigcup_{c\in\mathsf{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{lazy}} \Lambda^{\hat{\mathscr{E}}}_{\hat{\Psi}_c} \cup \Delta_{\hat{\Psi}_\mathsf{V}} \cup \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \Lambda_{\hat{\Psi}_\mathsf{v}}$, we conclude that

$$\begin{aligned}\delta \in\ & \Delta^{\mathsf{atom}(\mathsf{C})}_{\Pi} \cup \Delta^{\mathscr{E}}_{\Psi_\mathsf{C}} \cup \Delta_{\Psi_\mathsf{V}} \cup \Lambda^{(\mathsf{atom}(\mathsf{C})\setminus\mathsf{atom}(\Psi_\mathsf{C}))\cup(\mathscr{E}\setminus\mathsf{atom}(\Psi_\mathsf{V}))}_{\Pi\cup\Psi_\mathsf{V}\cup\Psi_\mathsf{C}} \cup \Delta^{\hat{\mathscr{E}}}_{\hat{\Psi}_\mathsf{C}} \\ & \cup \bigcup_{c\in\mathsf{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{lazy}} \Lambda^{\hat{\mathscr{E}}}_{\hat{\Psi}_c} \cup \Delta_{\hat{\Psi}_\mathsf{V}} \cup \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \Lambda_{\hat{\Psi}_\mathsf{v}}.\end{aligned}$$

By Lemma 6.1, we have $\Delta^{\mathsf{atom}(\mathsf{C})}_{\Pi} \cup \Delta^{\mathscr{E}}_{\Psi_\mathsf{C}} \cup \Delta^{\hat{\mathscr{E}}}_{\hat{\Psi}_\mathsf{C}} \cup \Delta_{\Psi_\mathsf{V}} \cup \Delta_{\hat{\Psi}_\mathsf{V}} = \Delta_{\Pi\cup\Psi}$, and by Lemma 6.4,

$$\begin{aligned}& \Lambda^{(\mathsf{atom}(\mathsf{C})\setminus\mathsf{atom}(\Psi_\mathsf{C}))\cup(\mathscr{E}\setminus\mathsf{atom}(\Psi_\mathsf{V}))}_{\Pi\cup\Psi_\mathsf{V}\cup\Psi_\mathsf{C}} \\ & \cup \bigcup_{c\in\mathsf{atom}(\mathsf{C}),\ \mathit{mode}(c)=\mathit{lazy}} \Lambda^{\hat{\mathscr{E}}}_{\hat{\Psi}_c} \\ & \cup \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \Lambda_{\hat{\Psi}_\mathsf{v}} \\ \subseteq\ & \Lambda_{\Pi\cup\Psi}.\end{aligned}$$

Hence, $\delta \in \Delta_{\Pi\cup\Psi} \cup \Lambda_{\Pi\cup\Psi}$. By Assumption 6.6, if $\delta \in \Lambda_{\Pi\cup\Psi}$, then $\delta \in \nabla_e$. In other words, δ is a member of the completion nogoods of $\Pi \cup \Psi$, or it was produced by UNFOUNDEDSETPROPAGATION under the eager setting. In any case, δ is also unit w.r.t. $\mathbf{A}_l \cap \mathbf{A}_e$ in the execution of PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$. Therefore, $\sigma \in \mathbf{A}_e$, contradicting the assumption.

In conclusion, we have $\mathbf{A}_l \subseteq \mathbf{A}_e$. Since PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$ returns in Line 12, so must PROPAGATION$(\Pi \cup \Psi_\mathsf{V} \cup \Psi_\mathsf{C}, \mathsf{P}, \mathsf{Q}, \nabla, \mathbf{A})$. □

A closer inspection of PROPAGATION shows that, if PROPAGATION returns in Line 3 under the eager setting and the assignment $\mathbf{A}$, then it also returns in Line 3 for every total assignment extending $\mathbf{A}$ under any setting, i.e., different from the eager setting.

Lemma 6.11

Let $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$ be a constraint program and $(\psi, \pi, \mathit{mode})$ be a setting. Let

$$\Psi_\mathsf{V} = \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{eager}} \psi(\mathsf{v}) \text{ and } \hat{\Psi}_\mathsf{V} = \bigcup_{\mathsf{v}\in\mathsf{V},\ \mathit{mode}(\mathsf{v})=\mathit{lazy}} \psi(\mathsf{v})$$

be the eager and lazy encodings of the variables' domains, and let

$$\Psi_{\mathsf{C}} = \bigcup_{c \in \text{atom}(\mathsf{C}),\ mode(c)=eager} \psi(c) \text{ and } \hat{\Psi}_{\mathsf{C}} = \bigcup_{c \in \text{atom}(\mathsf{C}),\ mode(c)=lazy} \psi(c)$$

be the eager and lazy encodings of the constraints, such that $\Psi = \Psi_{\mathsf{V}} \cup \hat{\Psi}_{\mathsf{V}} \cup \Psi_{\mathsf{C}} \cup \hat{\Psi}_{\mathsf{C}}$ as constructed in Line 1 of CDNL-LNG under the eager setting. Let P_{V} and P_{C} be the set of external propagators constructed in Lines 3 and 4 under the setting $(\psi, \pi, mode)$.

Let ∇ be a set of nogoods and $\mathbf{A}$ be an assignment such that PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$ returns in Line 3. Then, the execution of PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ returns in Line 3 for every assignment $\mathbf{A}_t \supseteq \mathbf{A}$ that is total.

Proof. Before we begin, let $\mathscr{E} \subseteq \text{atom}(\Psi_{\mathsf{V}} \cup \hat{\Psi}_{\mathsf{V}})$ be the externals of Ψ_c and let $\hat{\mathscr{E}} \subseteq \text{atom}(\Psi_{\mathsf{V}} \cup \hat{\Psi}_{\mathsf{V}})$ be the externals of $\hat{\Psi}_c$, where $c \in \text{atom}(\mathsf{C})$. Then, by construction, we have that P_{V} are the propagators that represent the set of nogoods $\Delta_{\hat{\Psi}_{\mathsf{V}}} \cup \bigcup_{\mathsf{v} \in \mathsf{V},\ mode(\mathsf{v})=lazy} \Lambda_{\hat{\Psi}_{\mathsf{v}}}$, and P_{C} are the propagators that represent the set of nogoods $\Delta^{\hat{\mathscr{E}}}_{\hat{\Psi}_{\mathsf{C}}} \cup \bigcup_{c \in \text{atom}(\mathsf{C}),\ mode(c)=lazy} \Lambda^{\hat{\mathscr{E}}}_{\hat{\Psi}_c}$.

Now, let PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$ return the pair $(\mathbf{A}_e, \nabla_e)$ in Line 3, and $\mathbf{A}_t \supseteq \mathbf{A}$ be a total assignment. We show that the execution of PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ returns in Line 3.

Since PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$ returns $(\mathbf{A}_e, \nabla_e)$ in Line 3, we have some nogood $\delta \in \Delta_{\Pi \cup \Psi} \cup \nabla_e$ such that $\delta \subseteq \mathbf{A}_e$, where ∇_e augments ∇ by loop nogoods from $\Lambda_{\Pi \cup \Psi}$. Hence, we distinguish the following three cases: $\delta \in \nabla$, $\delta \in \Delta_{\Pi \cup \Psi}$, or $\delta \in \Lambda_{\Pi \cup \Psi}$.

First, suppose $\delta \in \nabla$. Since $\delta \subseteq \mathbf{A}_e$ and $\mathbf{A}_e \subseteq \mathbf{A}_t$, we have that PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ returns in Line 3, trivially.

Second, suppose $\delta \in \Delta_{\Pi \cup \Psi}$. By Lemma 6.1, $\Delta_{\Pi \cup \Psi} = \Delta^{\text{atom}(\mathsf{C})}_{\Pi} \cup \Delta^{\mathscr{E}}_{\Psi_{\mathsf{C}}} \cup \Delta^{\hat{\mathscr{E}}}_{\hat{\Psi}_{\mathsf{C}}} \cup \Delta_{\Psi_{\mathsf{V}}} \cup \Delta_{\hat{\Psi}_{\mathsf{V}}}$. If $\delta \in \Delta^{\text{atom}(\mathsf{C})}_{\Pi} \cup \Delta^{\mathscr{E}}_{\Psi_{\mathsf{C}}} \cup \Delta_{\Psi_{\mathsf{V}}}$ then, again by Lemma 6.1, we have that $\delta \in \Delta^{(\text{atom}(\mathsf{C}) \setminus \text{atom}(\Psi_{\mathsf{C}})) \cup (\mathscr{E} \setminus \text{atom}(\Psi_{\mathsf{V}}))}_{\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}}$, and PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ returns in Line 3. On the other hand, if $\delta \in \Delta^{\hat{\mathscr{E}}}_{\hat{\Psi}_{\mathsf{C}}} \cup \Delta_{\hat{\Psi}_{\mathsf{V}}}$ then δ is in the set of nogoods represented by some propagator $\mathsf{p} \in \mathsf{P}_{\mathsf{V}} \cup \mathsf{P}_{\mathsf{C}}$. Recall that, by definition, an external propagator always returns a violated nogood if some of the nogoods it represents are violated w.r.t. a total assignment. Hence, there is some $\delta' \in \mathsf{p}(\mathbf{A}_t)$ such that $\delta' \subseteq \mathbf{A}_t$, and PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ returns in Line 3.

Finally, suppose $\delta \in \Lambda_{\Pi \cup \Psi}$. By Lemma 6.5, $\mathbf{A}$ is not unfounded-free for $\Pi \cup \Psi$. In turn, by Lemma 6.2, $\mathbf{A}$ is not unfounded-free for Π or $\mathbf{A}$ is not unfounded-free

for Ψ. Then, by Lemma 6.5 again, we have that there is some nogood $\delta' \in \Lambda_{\Pi}^{\text{atom}(\mathsf{C})} \cup \Lambda_{\Psi}$ such that $\delta' \in \mathbf{A}$. Alternating between Lemma 6.2 and Lemma 6.5 we can proceed with dissecting the sets of loop nogoods, and conclude that there is some $\delta'' \in \Lambda_{\Pi}^{\text{atom}(\mathsf{C})} \cup \Lambda_{\Psi_{\mathsf{C}}}^{\mathscr{E}} \cup \bigcup_{c \in \text{atom}(\mathsf{C}),\ mode(c)=lazy} \Lambda_{\hat{\Psi}_c}^{\hat{\mathscr{E}}} \cup \Lambda_{\Psi_{\mathsf{V}}} \cup \bigcup_{\mathsf{v} \in \mathsf{V},\ mode(\mathsf{v})=lazy} \Lambda_{\hat{\Psi}_{\mathsf{v}}}$ such that $\delta'' \in \mathbf{A}$. We make yet another case destinction. First up, if $\delta'' \in \Lambda_{\Pi}^{\text{atom}(\mathsf{C})} \cup \Lambda_{\Psi_{\mathsf{C}}}^{\mathscr{E}} \cup \Lambda_{\Psi_{\mathsf{V}}}$ then, by Lemma 6.4, we have that $\delta'' \in \Lambda_{\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}}^{(\text{atom}(\mathsf{C}) \setminus \text{atom}(\Psi_{\mathsf{C}})) \cup (\mathscr{E} \setminus \text{atom}(\Psi_{\mathsf{V}}))}$, and PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ returns in Line 3, since UNFOUNDEDSETPROPAGATION adds δ'' to the recorded nogoods in Line 11 via Line 13. On the other hand, if $\delta'' \in \bigcup_{c \in \text{atom}(\mathsf{C}),\ mode(c)=lazy} \Lambda_{\hat{\Psi}_c}^{\hat{\mathscr{E}}} \cup \bigcup_{\mathsf{v} \in \mathsf{V},\ mode(\mathsf{v})=lazy} \Lambda_{\hat{\Psi}_{\mathsf{v}}}$ then δ'' is in the set of nogoods represented by some propagator $\mathsf{p} \in \mathsf{P}_{\mathsf{V}} \cup \mathsf{P}_{\mathsf{C}}$. Recall that, by definition, an external propagator always returns a violated nogood if some of the nogoods it represents are violated w.r.t. a total assignment. Hence, there is some $\delta''' \in \mathsf{p}(\mathbf{A}_t)$ such that $\delta''' \subseteq \mathbf{A}_t$ that is added to the recorded nogoods in Lines 5 or 8 via Line 13, and PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ returns in Line 3.

Hence, if PROPAGATION$(\Pi \cup \Psi, \emptyset, \emptyset, \nabla, \mathbf{A})$ returns in Line 3, so does PROPAGATION$(\Pi \cup \Psi_{\mathsf{V}} \cup \Psi_{\mathsf{C}}, \mathsf{P}_{\mathsf{V}}, \mathsf{P}_{\mathsf{C}}, \nabla, \mathbf{A}_t)$ for every total assignment $\mathbf{A}_t \supseteq \mathbf{A}$. □

We now show that, given the above, soundness and termination of CDNL-LNG follows for any setting.

Theorem 6.12: Soundness and Termination of CDNL-LNG

Let $\mathbb{P}$ be a constraint program and $(\psi, \pi, mode)$ any setting. CDNL-LNG terminates, and returns a constraint answer set of $\mathbb{P}$ if and only if some constraint answer set of $\mathbb{P}$ exists.

Proof. Let $(\psi, \pi, mode)$ be any setting. We begin with showing that CDNL-LNG is sound under $(\psi, \pi, mode)$. We have to consider the following two cases: The first one is that CDNL-LNG returns a pair $(\mathbf{A}^{\mathbf{T}} \cap \text{atom}(\Pi), \text{assign}_{\mathsf{V},\mathsf{D}}(\mathbf{A}))$ in Line 17, and the second one is that CDNL-LNG returns *no constraint answer set* in Line 11.

First, suppose CDNL-LNG under the setting $(\psi, \pi, mode)$ returns the pair $(\mathbf{A}^{\mathbf{T}} \cap \text{atom}(\Pi), \text{assign}_{\mathsf{V},\mathsf{D}}(\mathbf{A}))$ in Line 17. Then, Lemma 6.10 guarantees that PROPAGATION under the eager setting draws the same conclusions. Since the test in Line 16 is passed, $\mathbf{A}$ is a total assignment. Furthermore, since the test in Line 10 is passed and by Lemma 6.11, $\mathbf{A}$ is not conflicting with the eager encoding. By Theorem 6.9, $(\mathbf{A}^{\mathbf{T}} \cap \text{atom}(\Pi), \text{assign}_{\mathsf{V},\mathsf{D}}(\mathbf{A}))$ is a constraint answer set.

Secondly, suppose CDNL-LNG under the setting $(\psi, \pi, mode)$ returns *no constraint answer set* in Line 11. Then, Lemma 6.10 guarantees that PROPAGATION

under the eager setting draws the same conclusions. By Theorem 6.9, $\mathbb{P}$ has no answer set.

Termination of CDNL-LNG follows from the termination of CDNL-ASP (Gebser, 2011), and CDNL (Zhang and Malik, 2003) in particular. □

Note that, since we have abstracted external propagators from any specific implementation, in practice, the termination of CDNL-LNG is dependent on the termination of the actual algorithms underlying external propagation. Naturally, complete algorithms have to be employed whenever complete results are desired.

Also note that CDNL-LNG can be made into an enumeration algorithm, e.g., following Gebser et al. (2007c).

6.4 Experimental Results

For its empirical assessment, we have implemented LNG for finite domain variables, the ALL-DIFFERENT and integer LINEAR constraints within our prototypical CASP system *inca*[10]. It is based on a development version of the ASP system *clingo* (3.0.92)[11], and uses CDNL-LNG as its core reasoning engine that it inherits from *clasp* (1.3.10)[11]. This includes conflict-driven learning, lookback-based search heuristics, deletion strategies, restart policies, watched literals, solution enumeration, optimisation, native handling of cardinality constraint rules, and so on (Gebser et al., 2007b,c).

The architecture of *inca* is illustrated in Figure 6.5. Given a constraint program $\mathbb{P} = (\mathsf{V}, \mathsf{D}, \mathsf{C}, \Pi)$, its PREPROCESSOR creates all necessary objects for CDNL-LNG: It computes the strongly connected components of Π, and eagerly generates all completion nogoods of Π and those constraints in C and variables in V that were configured to be represented eagerly. In turn, it initialises the external propagators for the remaining CP constructs, i.e., those constraints in C and variables in V that were configured to be represented lazily. Search and propagation is controlled by *clasp*'s ASP decision engine. Though, as we have outlined in Algorithm 6.4, PROPAGATION distinguishes between ASP inference and external propagation. Whilst ASP inference is performed by *clasp*, it defers the handling of all remaining CP constructs to external propagators. To facilitate the latter, the ASP decision engine communicates its current assignment $\mathbf{A}$ to the constraint propagation engine which extracts the updated domains of the constraint

[10] `http://potassco.sourceforge.net/labs.html`
[11] `http://potassco.sourceforge.net/`

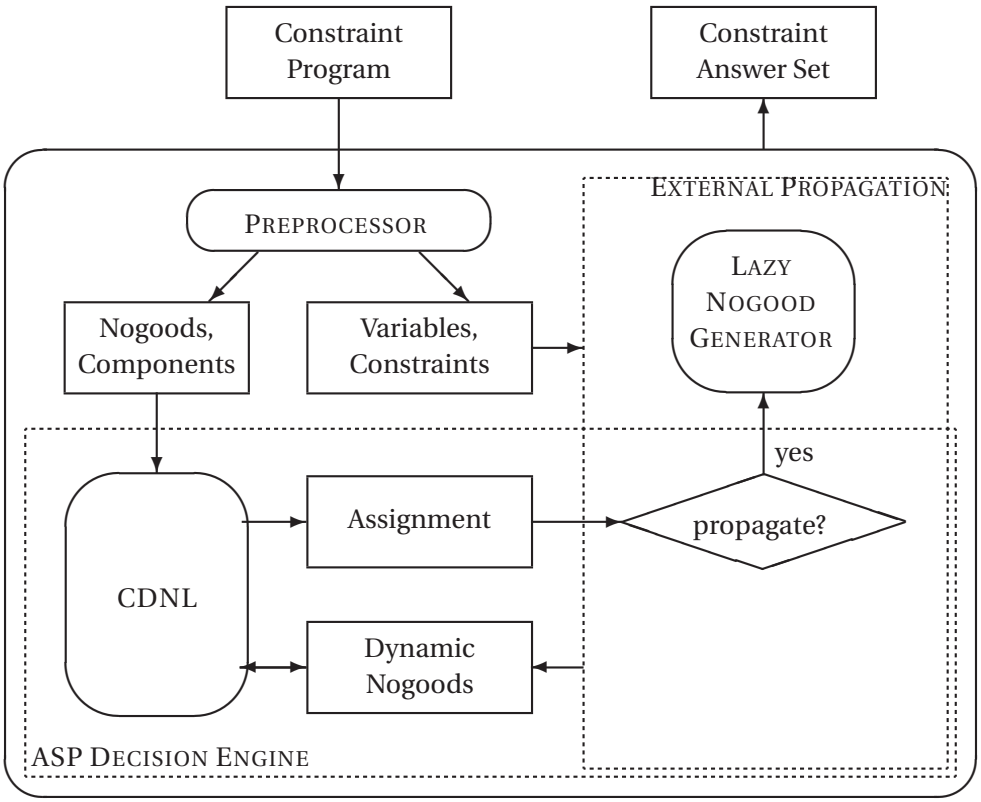


Figure 6.5: Architecture of *inca*.

variables and the state of the constraint atoms. This is implemented via watched literals (Moskewicz et al., 2001; Gent, 2013). If some external propagator p reports inference, it reports back a lazily generated nogood from p(**A**).

The default setting of *inca* uses an ALL-DIFFERENT propagator that enforces arc consistency, while $inca^{DC}$ enforces domain consistency, representing an infeasible ASP encoding of ALL-DIFFERENT, i.e., one based on all Hall sets. Our implementation of the external propagators representing the conditions induced by ALL-DIFFERENT constraints use a separate low priority queue, staged propagation, incremental matching, and incremental exploitation of strongly connected components (cf. Gent et al., 2008).

We include the CASP system *clingcon* (2.0.0-beta)[11] in our analysis. It also extends *clingo*, but integrates the CP solver *gecode* (3.7.1)[12]. Recall that, similar to our approach, *clingcon* is based on CDNL and abstracts from the constraints via constraint atoms. Following the idea of SMT, however, *clingcon* employs its

[12] `http://www.gecode.org/`

CP solver to check the existence of a variable assignment that does not violate any constraint (according to the assignment to constraint atoms). Compared to its predecessor that we have previously evaluated in Section 5.1, this newer version of *clingcon* encodes the state of the CP solver into nogoods over constraint atoms, whenever *gecode* yields a conflict or determines the truth value of a constraint atom. This process can be seen as a very limited form of LNG, though, the CP solver does not contribute any information about its propagation to the CONFLICTANALYSIS process. Hence, this approach may also require exponentially more search than LNG as it cannot access the state of the propagator, e.g., via to intermediate atoms.

We have set *clingcon* to generate nogoods by looking at the implication dependency between constraint atoms according to the *irreducibly inconsistent set construction* method in *forward* mode (Ostrowski and Schaub, 2012), when we noticed that this option significantly improves the performance of *clingcon* on our benchmarks. In general, however, this indirect method incurs some overhead for the CONFLICTANALYSIS process. Moreover, recall that *clingcon* applies backtracking search through *gecode* following some variable and value selection heuristic to determine an assignment to the variables, while in *inca* all search is carried out through the CDNL-LNG engine. Another, more subtle difference to *inca* is that *clingcon* prefers unfounded set propagation over constraint propagation. As with *inca*, the default setting of *clingcon* uses an ALL-DIFFERENT propagator that enforces arc consistency, while $clingcon^{DC}$ enforces domain consistency.

For a comparison with the state-of-the-art in answer set solving, our experiments also consider eager encodings generated by *inca*. Since eager encodings only rely on *inca*'s ASP subsystem *clingo* (3.0.92) for the solving process, we denote this option by *clingo*. Note that the loop nogoods representing unfounded set inference are encoded lazily in all settings. Experiments were run on a 2.00 GHz PC under Linux, where each run was limited to 600 s time and 2 GByte RAM.

Quasigroup Completion

We first reconsider quasigroup completion problems (QCPs) to test LNG for the ALL-DIFFERENT constraint. Previous experiments on this benchmark domain with $n = 20$, reported in Section 5.1, have shown eager encodings outperforming hybrid CASP systems, including a preliminary version of *clingcon*.

In order to study the computational behaviour on large-scale tables we have increased size to $n = 40$. This quadruples the amount of variables involved, and

doubles the size of their domains. We have generated 200 instances near to the easy-hard-easy phase transition (Cheeseman et al., 1991). The phase transition can be observed, e.g., at the maximum memory consumption of *clingo* in relation to the preassignment rate. In fact, we observe most cases of memory exhaustion with a preassignment rate of 50 percent. The graph in Figure 6.6 (top) illustrates the maximum memory usage of *clingo* and *inca* in GBytes by preassigned values in percent.

Since all atoms in the CASP encoding for QCP are constraint atoms whose truth value is known a-priori, *clingcon* cannot make use of its conflict-driven capacities. Instead, search and propagation is carried out by its backtracking-based CP solver, i.e., following a smallest domain and smallest value first selection heuristic, whilst search in *clingo* and *inca* is based on activity in CONFLICTANALYSIS. This is likely be the reason that renders *clingcon* ineffective on this benchmark domain, solving only 9 of 200 instances. It is therefore excluded from Figure 6.6. To our surprise, enabling learning capacities in *clingcon* by decomposing the ALL-DIFFERENT constraints into primitive constraints, like in the eager encoding used by *clingo*, exhausts CPU time on all instances.

As can be seen in Figure 6.6 (bottom) which gives the run time in minutes against the numbers of solved instances, within the allowed execution time, *clingo* solves almost half of all instances. Though, *clingo* performs drastically worse on QCPs with the increased size, compared to previous experiments.

The use of LNG in *inca*, on the other hand, avoids huge encodings while preserving conflict-driven learning capacities. In fact, *inca* solves twice as many instances as *clingo* within one hour of execution time, and nearly twice as many instances overall.

Propagation that achieves domain consistency further improves performance. Both CASP systems, *clingcon* and *inca* benefit thereof, as results on QCPs demonstrate (see Table 6.3). This emphasises the importance of external propagation, as achieving domain consistency is impossible via eager encodings in general.

Quasigroup Existence

We also reconsider quasigroup existence problems (QEPs), i.e., the problem of determining the existence of certain classes of quasigroups with properties QG1–GG7 (see Section 5.1) which are modelled using primitive constraints. This weakens the overall consistency achieved by propagation.

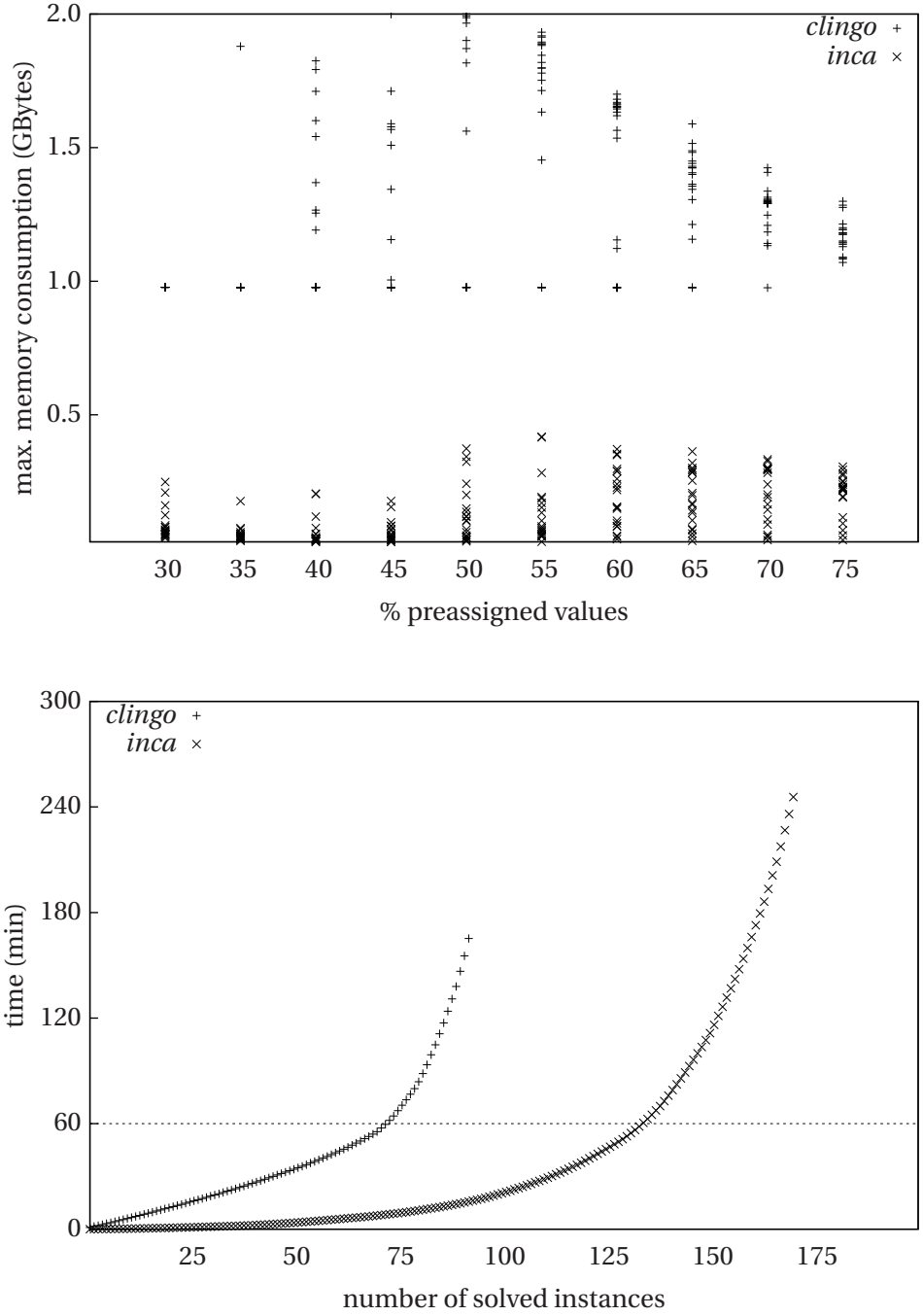


Figure 6.6: Results on QCPs of size 40.

Table 6.1: Run time results in seconds for QEPs.

	n	*clingo*	*clingcon*	*clingcon*^{DC}	*inca*	*inca*^{DC}
	7	**1.5**	6.8	7.9	**1.5**	**1.5**
QG1	8	8.6	42.8	176.2	13.4	**4.4**
	9	—	—	—	**450.1**	—
	7	**1.5**	9.2	6.9	**1.5**	**1.5**
QG2	8	26.9	—	—	**7.4**	19.0
	9	—	—	—	69.7	**61.2**
	7	**0.1**	0.6	0.3	**0.1**	**0.1**
QG3	8	**0.1**	**0.1**	1.1	**0.1**	**0.1**
	9	**13.3**	—	—	18.4	16.0
	7	**0.1**	0.6	0.6	**0.1**	**0.1**
QG4	8	**0.2**	11.8	17.0	**0.2**	0.3
	9	0.3	24.6	22.4	**0.1**	0.6
	12	**1.7**	—	—	**1.7**	3.2
QG5	13	**33.3**	—	—	41.6	70.1
	14	**155.4**	—	—	217.8	219.4
	10	0.3	19.6	27.6	**0.2**	**0.2**
QG6	11	1.0	488.0	442.7	**0.8**	1.4
	12	**14.7**	—	—	42.9	14.9
	10	5.5	—	267.1	8.0	**1.3**
QG7	11	197.7	—	—	327.3	**9.2**
	12	—	—	—	—	**107.5**

In comparison to our previous experiments with *clingcon* (0.1.2), that had only very limited capacities for conflict-driven learning involving constraint atoms and no support for global constraints like the ALL-DIFFERENT constraint, *clingcon*'s newer version improves run time. Still, it cannot compete with the translation-based approach represented by *clingo*. Our results shown in Table 6.1 demonstrate that techniques which encode CP constructs into nogoods, continue to benefit from learning constraint interdependencies a lot more than the hybrid system. Whilst *inca* competes with *clingo*, it outperforms *clingo* on the hardest instances.

Graceful Graphs

To test LNG for a combination of ALL-DIFFERENT and integer LINEAR constraints, we reconsider instances of the graceful graph problem (GGPs; see Section 5.1), in particular, the problem of determining the existence of a graceful labelling of

Table 6.2: Run time results in seconds for GGP.

n	*clingo*	*clingcon*	*clingcon*^{DC}	*inca*	*inca*^{DC}
3	3.5	1.1	**0.4**	3.8	5.6
4	**0.1**	0.9	0.8	**0.1**	0.2
5	**0.3**	3.8	1.3	3.8	**0.3**
6	**0.2**	57.1	25.9	1.2	0.3
7	**0.7**	—	12.5	7.2	2.8
8	**0.9**	—	22.3	15.3	3.0
9	**1.6**	—	156.1	—	29.2
10	**5.4**	—	—	—	31.3
11	**14.5**	—	—	—	28.8
12	—	—	—	—	**24.8**
13	—	—	—	—	**132.0**
14	—	—	—	—	—

double-wheel graphs. The results shown in Table 6.2 confirm previous observations. Moreover, the eager encoding represented by *clingo* competes with LNG on this benchmark domain even if strong propagators are employed. Though, this observation does not hold for larger graphs any more.

Packing

To further test LNG for integer LINEAR constraints, we consider instances of the *Packing* problem, that is the problem of packing objects together inside a container. A special case of the packing problem formed a benchmark class in the third ASP competition (Calimeri et al., 2011). There, a set of squares of known dimensions had to be packed into a rectangular area such that no two squares overlap each other and all the squares are packed, possibly leaving space in the rectangular area unoccupied. Problems like packing are particularly hard to solve with ASP systems because they typically involve large domains that affect the size of their encoding. In fact, the encoding that was given in the system track of the competition quickly reaches the memory limit of 2 GBytes in 49 out of 50 instances, while the CASP systems *clingcon* and *inca* solve every instance within a reasonable amount of space and time. On the Packing domain, the advantage of *inca* over *clingcon* is only marginal.

Table 6.3: Average run time in seconds over completed runs on QCP, QEP, GGP, Packing, and Numbrix benchmarks. Number of completed runs are given in parenthesis.

Benchmark class	*clingo*	*clingcon*	*clingcon*^{DC}	*inca*	*inca*^{DC}
Quasigroup Completion (200)	106.6 (93)	34.4 (9)	**4.6** (**200**)	86.2 (171)	24.7 (**200**)
Quasigroup Existence (21)	25.7 (18)	61.4 (10)	88.2 (11)	60.3 (**20**)	**26.6** (**20**)
Graceful Graphs (10)	3.0 (9)	15.7 (4)	31.3 (7)	5.2 (6)	**12.6** (**10**)
Packing (50)	104.1 (1)	33.1 (**50**)	33.1 (**50**)	**24.6** (**50**)	**24.6** (**50**)
Numbrix (12)	10.4 (**12**)	17.4 (**12**)	51.3 (**12**)	**1.3** (**12**)	5.2 (**12**)
weighted, penalised time	228.3	267.0	124.6	103.1	**24.2**

Numbrix

Numbrix is a variant of Hidato, the number-placement game we have described in Chapter 1. It is played on an $n \times n$ grid. Similar to Hidato, its objective is to fill the grid with consecutive integers that connect horizontally and vertically, but, different from Hidato, not diagonally. Another difference is that the positions of the smallest and the highest number are not always given. Still, every Numbrix puzzle should have a unique solution.

Our experiments determine the existence of a unique solution for puzzles that stem from Marilyn vos Savant's column in the *Parade* magazine[13] with size $n = 9$. While *clingo*, *clingcon*, and *inca*, all are able to solve all puzzles considered in our analysis, we observe that execution time improves when CP constructs are encoded into nogoods, as in the options *clingo* and *inca*, and drastically improves when this encoding is lazy, as in the option *inca*.

Summary of Results

A summary of our experiments is provided in Table 6.3. It also gives the average run time over all benchmark classes, where each benchmark class is weighted equally, and penalising timeouts with the time limit. We can draw a few interesting conclusions.

[13] `http://www.parade.com/`

First, execution time can improve when CP constructs are treated by external propagation rather than encoding them eagerly. The latter can lead to huge encodings, in particular, when large domains are involved.

Second, the advantage of generating nogoods to describe the inferences of constraint propagators is that CDNL can exploit constraint interdependencies for directing search, and most importantly CONFLICTANALYSIS. The fact that *clingcon* does not encode CP constructs into nogoods, by design, is likely to be the reason for its limited success in our experiments, where *clingcon* is particularly ineffective on quasigroup problems.

Third, experiments show that our approach, represented through *inca*, combines the best of both worlds: It can avoid huge encodings via abstraction to external propagation while retaining the ability to make the encoding explicit. It outperforms the state-of-the-art in CASP solving on individual benchmark classes, and is more robust over all benchmark instances.

On most benchmarks, lazy generation of infeasible ALL-DIFFERENT encodings via external propagation has further increased performance, where $inca^{DC}$ improves on *clingo* by one order of magnitude.

The Fourth Answer Set Programming Competition

In order to introduce our techniques to a competitive environment and increase their visibility, we entered our CASP system *inca* into the fourth ASP competition[14] (Alviano et al., 2013a). Participants were competing on a collection of benchmark problems selected via a peer review process. Benchmarks stem from a variety of domains, including real world applications, and two of our contributions: instances from graceful graphs and connected maximum-density still life problems. We have also submitted a package to the Model & Solve track of the competition. The Model & Solve track was oriented towards developers of systems for declarative problem solving to showcase their solvers on 15 problem domains. Our submission package included our CASP system *inca* along with a CASP specification and a solver setting for each benchmark. While the set of benchmarks was known to the participants, all submissions were compared on a set of undisclosed instances. Amongst others, the following benchmarks were considered:

Permutation Pattern Matching Given a permutation text and a permutation pattern, the *permutation pattern matching* problem (PPM) is to determine the

[14] `http://www.mat.unical.it/aspcomp2013/`

existence of a sub-permutation in the text that has the same relative order as specified by the pattern.

Valves Location The *valves location* problem (Valve) consists of placing a given number of valves in a water distribution network. The objective is, in case of damage of a pipe, to minimise the disruption caused by isolating the affected subnetwork.

Graceful Graphs Recall, the graceful graph problem (GGP) is to determine the existence of a graceful labelling of a graph.

Bottle Filling Given a rectangular grid, a placement of convex containers (e.g., bottles) on the grid that may hold liquid, and the sum of cells in any container along each row and column that are filled with liquid. The *bottle filling* problem (Fill) is to determine which cells are filled, and which ones are empty, under consideration of gravity.

Nomystery Given a graph with integer-labelled edges and a truck that can move along its edges, load and unload packages at vertices. Each move consumes fuel equal to the edge label. The task in *nomystery* (Nom) is to find a sequence of actions to move packages between vertices with a limited amount of fuel.

Ricochet Robots This board game is played on a grid with walls that separate some adjacent cells. A set of robots with distinct colours is placed on predefined cells. (A cell may hold at most one robot.) Each robot can move horizontally or vertically in any direction, but does not stop until it encounters a wall or another robot. The task is to find a sequence of actions to move a designated robot to some target cell within a given number of moves. The *ricochet robots* (RiRo) benchmark is further described in (Gebser et al., 2013a).

Crossing Minimisation A hierarchical graph is a type of graph layout in which the vertices are organised in horizontal layers with the edges directed downwards. Naturally, crossing of edges that connect vertices from the same two layers occur. The objective in the *crossing minimisation* problem (Cross) is to minimise the number of crossings by re-arranging the vertices in each layer of a given hierarchical graph.

Weighted-sequence The *weighted-sequence* problem (WSeq; Lierler et al., 2012) is a variant of the problem of finding a minimal-cost join order in query optimisation. Given a set of vertices, where each vertex is associated with two costs (weight and cardinality), the task is to find a permutation and a colouring of the vertices such that the cost (according to a given function defined on colour, weight, and cardinality of neighbouring vertices) is below or equal the given maximum.

Incremental Scheduling Motivated by industrial printing, the *incremental scheduling* problem (Sched; Balduccini, 2011) is an incremental version of the job scheduling problem, where a schedule has to be updated w.r.t. the addition of jobs and devices going offline.

Whilst the above problems are naturally modelled with CASP, the remaining problems in the Model & Solve track of the competition, i.e., *connected maximum-density still life*, *sokoban*, *solitaire*, and *strategic companies*, do not benefit from the increased modelling convenience at this stage, or are naturally solved using polynomial algorithms, i.e., (polynomial) *reachability* and *stable marriage*.

Competing participants submitted packages that include ASP and CASP systems, and related solvers that extend logic programming to integrate higher-level constructs. On above benchmarks, the *potassco* (Gebser et al., 2011c) team employed their ASP systems *gringo* and *clasp* (2.1.2), but did not opt to use their CASP system *clingcon*. The *ezcsp* team, on the other hand, selected *clingcon* (2.0.3) on weighted-sequence and incremental scheduling problems, and chose between the ASP systems *dlv*, *clingo* (3.0.3), and *clingof* (0.4.3) for the remaining benchmarks. The latter is a variant of *clingo* that supports non-Herbrand functions (Balduccini, 2013), but it was only used on the nomystery benchmark. Observe that *ezcsp*'s eponymous CASP system was not submitted to the competition. In contrast, the *idp* team contributed *idp* (3.1.4), their model expansion system for an extension of classical logic (Wittocx et al., 2008). It implements lazy clause generation, a technique related to LNG (see de Cat et al., 2015), and handles CP constructs in aformentioned benchmarks, except for bottle filling, nomystery, and ricochet robots. To complete the list of participants, the *b-prolog* (Zhou, 2012) team's system integrates dynamic programming, CP solving, and compilation to SAT, whilst the *enfragmo* (Aavani et al., 2012) team committed to the translation-based approach, transforming each benchmark's instance into a SAT problem.

Table 6.4: Results from the Fourth Answer Set Programming Competition.

System	Total	PPM	Valve	GGP	Fill	Nom	RiRo	Cross	WSeq	Sched
potassco	1126 (15)	98	58	84	95	51	79	79	100	63
inca	832 (13)	99	24	51	94	98	46	43	0	0
ezcsp	769 (14)	67	23	33	94	0	40	23	87	58
idp	534 (10)	96	0	35	64	78	0	19	0	6
b-prolog	284 (6)	100	16	43	N/A	100	12	13	0	0
enfragmo	66 (4)	24	N/A	3	33	0	N/A	N/A	0	N/A

We have extracted the competition's total results[14] in points, with detailed results for some benchmarks in Table 6.4. The point system is explained in (Alviano et al., 2013a). It considers run time and quality of a solution to each benchmark instance, where at most 100 points could be achieved on each problem domain. An incorrect output, e.g., due to an error in the model or the solver, was penalised with no points for the entire benchmark. This is also reflected in Table 6.4 by a decrement in the number of valid submissions in parenthesis (maximum is 15). Unfortunately, our team was affected twice. First, a modelling error was too strict on the lower bound of a variables' domain in some instances of the weighted-sequence problem, and second, a software bug in *inca*'s subsystem *clingo* triggered by some instances of the incremental scheduling problem. (At closer inspection, the instances that remained unaffected by either problem show *inca* competing with the approaches of *potassco* and *ezcsp*, i.e., *clasp* and *clingcon*, respectively.) Still, *inca* was placed ahead of the majority of the contestants, including all hybrid systems, winning second place.

It is difficult, if not impossible, however, to draw any meaningful conclusion from the results of the competition. The reason is that the problem encodings submitted with each system were not restricted to (syntactic adjustments of) a common model, e.g., the ASP encoding provided by the organising committee with each benchmark. This has encouraged participants to investigate and encode special- and corner cases of each problem domain as much as resources were available to each contestant. As the *potassco* team has shown spectacularly, finding a tailor-made encoding for a specific problem and algorithms can drastically prune search, and provide an advantage even over algorithms that are more naturally suited. While this is represents a remarkable achievement, for our purposes, a fair comparison between systems must undoubtedly be based on a shared model.

That aside, there is no single contestant that performs best on every bench-

mark, and overall, the approaches by *potassco*, *inca*, and *ezcsp* are most mature w.r.t. the number of problem domains successfully tackled. This may be traced back to *clasp*'s implementation of CDNL that is shared amonst their systems, including *clingo*, *clingcon*, and *inca*.

6.5 Related Work

Related work on separating the conditions on the answer sets of a program induced by sub-programs, or for this purpose, *modules*, is plentiful. Our theoretical underpinnings are founded on the modularity offered through the splitting set theorem by Lifschitz and Turner (1994). Oikarinen and Janhunen (2006) proposed the *module theorem* which strengthens the splitting set theorem by allowing negative recursions between modules. In turn, Oikarinen and Janhunen's modules slightly generalise programs with externals.

Example 6.10

Consider the program Π_1 with externals over $\{a\}$ and the program Π_2 with externals over $\{b\}$, where

$$\Pi_1 = \left\{ \ a \leftarrow not\ b \ \right\} \text{ and } \Pi_2 = \left\{ \ b \leftarrow not\ a \ \right\}.$$

The splitting set theorem does not allow for the join $\Pi_1 \cup \Pi_2$ as there exists no splitting set separating the joint program into Π_1 and Π_2. In other words, there is a recursive dependency involving atoms from both programs.

Without introducing the definitions needed to fully capture the join operation of modules, the module theorem, on the other hand, allows for the join as there is no *positive* dependency between atoms from Π_1 and Π_2.

When the module theorem is combined with the results from Gebser et al. (2007a), i.e., the inferences from the completion and loop formulas of a module is captured by nogoods, then the proofs of our Lemmas 6.1 and 6.4, and Theorem 6.6 from Section 6.1 will follow straightforwardly. In our setting, however, there is no practical relevance of admitting negative recursion between modules as we represent each constraint by a program with externals representing only the variables's domains. Any recursive dependency between constraints is always expressed in the underlying constraint logic program (with externals representing the constraints).

Further abstraction, away from any particular syntax and semantics, was investigated by Järvisalo et al. (2009). The module theorem for disjunctive logic pro-

grams is presented in (Janhunen et al., 2009).

Whilst the aforementioned work mostly focused on semantic aspects of modularisation, including join operations between modules, there have also been quite many substantial applications, e.g., in *incremental* (Gebser et al., 2008), *reactive* (Gebser et al., 2011a), and *time-decaying* (Gebser et al., 2012a) answer set solving.

Our centre point here were low-level computational, nogood-based alternatives for representing the conditions on the overall semantics (i.e., the answer sets) induced by a module, including lazy generated nogoods by an external propagator. Of particular interest to us are their guarantees in terms of inference. For instance, we can now characterise the implementation of FL inference by Gebser et al. (2007a) as conflict-optimal external propagation of the loop nogoods of a program. An extension that also considers BL inference, e.g., via our methods presented in Chapter 3, can achieve inference-optimal external propagation. A very recent study conducted by Lierler and Truszczynski (2014) looked at building hybrid inference systems. Their methods, however, are too high-level to capture the propagation within modules.

The direction of research in the field of ASP closest to our notion of external propagation is HEX-programming (Eiter et al., 2009). It integrates ASP with external sources of computation. Although they have been tightly incorporated with conflict-driven answer set solving in a study by Eiter et al. (2012), external propagators are treated as black-boxes that do not contribute any information about their inference to the CONFLICTANALYSIS procedure. As a workaround, Eiter et al. apply indirect measures to learn nogoods from observing the input-output relationship of external sources (uninformed learning), with few optimisations if the relationship is known to be monotonic or functional (informed learning). This can be regarded as a very limited form of LNG.

LNG is motivated by lazy clause generation (Ohrimenko et al., 2009), a SAT-based approach to solving CSP where lazy clause generators encode the inference of constraint propagation rules into clauses. Hence, exploiting the close relationship between ASP and SAT, an obvious alternative to integrating CP via LNG with answer set solving is the integration of ASP into a lazy clause generation CP system. Aziz et al. (2013) made the effort to incorporate FL inference with their lazy clause generation solver. As they point out, their system is close to what we have implemented. However, LNG is fundamentally more general than lazy clause generation: Every external propagator can be made to simulate lazy clause generation by syntactically representing every nogood by a clause. But, in principle, other

ASP constructs are also possible, such as cardinality constraint rules, their generalisation to weight constraint rules (Niemelä et al., 1999), and aggregations and other forms of set constructions (Pelov, 2004; Faber et al., 2011). Elkabani et al. (2004) provide a generic framework which provides an elegant treatment of such extensions to ASP, employing constraint propagators for their handling, though, without support for conflict-driven techniques. A thorough approach to incorporating a native treatment of weight constraint rules within a conflict-driven framework is presented in (Gebser et al., 2009b). As such constructs in turn represent sets of nogoods, we can use them to implement staging mechanisms, e.g., by giving preference to the handling of native ASP constructs over external propagation. Another difference to lazy clause generation is that the SAT-based approach also disregards the negation of a constraint, i.e., the truth value of each constraint atom is known a-priori. In our approach, every constraint c is reified via constraint atoms $[\![c]\!]$ (or $[\![\overline{c}]\!]$). There is, however, no technical limitation to lazy clause generation that should impede on the integration of propagators for the negation of a constraint.

A paradigm related to CASP that has very recently seen the integration of lazy clause generation into a conflict-driven decision procedure is represented through *idp* (de Cat et al., 2015, ; cf. Section 6.4). In their experimental analysis, de Cat et al. report significant speed-ups over eager encodings. Although *idp* and *inca* implement similar techniques, their system does not seem to perform on a par with ours, even on similar models, given the results from the fourth ASP competition.

In the field of CASP, the line of research closest to ours is the one of Gebser et al. (2009c), that we have discussed repeatedly in previous chapters of this thesis. Following a hybrid approach to constraint answer set solving, Gebser et al. incorporated a CP system into a conflict-driven ASP solver that interleaves search and ASP inference with constraint propagation. In accordance with the study conducted by Balduccini and Lierler (2013) on integration schemes, Gebser et al. follow a clear-box approach. Though, their approach still treats the CP solver as a black-box which does not contribute any information about its propagation, e.g., to the ConflictAnalysis process. As a workaround, recent advances reported in (Ostrowski and Schaub, 2012) apply indirect measures to learn from the underlying CSP by looking at the implication graph between constraint atoms. In contrast to Ostrowski and Schaub indirect method, our approach can make the inference of constraint propagation fully transparent. An implementation of Ostrowski and Schaub's technique is given through the CASP system *clingcon*, to which we

have compared our approach in a range of experiments. Our analysis has shown that *clingcon* competes with our solution on many benchmarks. On some experiments, however, their indirect method of integrating information from external propagation limits the exploration of constraint interdependencies. This is when our approaches outperform *clingcon*.

Alternative computation models for ASP and CASP have also been proposed, for instance, aimed at limiting the need for preliminary grounding (Lefèvre and Nicolas, 2009; Dal Palù et al., 2009, e.g.), or based on alternative problem solving paradigms (Liu et al., 2012, e.g.).

6.6 Conclusions

We presented a comprehensive extension for answer set solving to address the scalability and efficiency of ASP, called LNG, a technique motivated by lazy clause generation (Ohrimenko et al., 2009) for solving CSP. Founded on a nogood-based characterisation of external propagation, LNG allows for representing encodings that are otherwise impractical. This has been demonstrated with an ASP encoding of ALL-DIFFERENT that has to consider a worst-case exponential number of Hall sets. However, external propagators can make parts of the encoding explicit via lazily generated nogoods whenever conflict or inference is triggered. Recording these nogoods can avoid their re-computation, where deletion strategies may be applied to control their amount.

We started with theoretical underpinnings of our approach, centred around variants of the splitting set theorem (Lifschitz and Turner, 1994). They add a form of modularity that allows a programmer to mix constructs and processing techniques through eager and lazy representations. To characterise the effect of UP on nogoods generated by external propagators, we have introduced the notions of conflict- and inference optimality. A prominent example of conflict-optimal external propagation of a program's loop nogoods was given by Gebser et al.'s implementation of FL inference. We have argued that it can be made inference-optimal by considering the effects of BL inference.

We seamlessly applied our techniques to constraint answer set solving by employing constraint propagation algorithms as external propagators. The individual algorithms can be drawn from CP, a research area that is largely concerned with efficient propagation in solving CSP. Whilst convenient at first glance, existing implementations of constraint propagation do not encode their inference

in form of lazy nogoods. This may represent a significant obstacle to LNG, as every CP construct has to be addressed anew and made into an external propagator. Where this limitation can be overcome, LNG represents a significant advance in the state-of-the-art of integrating information from constraint propagation with CDNL. In contrast to hybrid CASP systems that must employ indirect measures by looking at inference graphs or input-output relationships of external sources, our approach can make the inference of constraint propagation fully transparent.

Finally, we presented our key algorithm, called CDNL-LNG. It inherits many advantages from translation-based constraint answer set solving. In particular, constraint propagation is made transparent trough a joint nogood-based model, sharing the representation of the variables' domains between encodings. Hence, CDNL-LNG maintains the outstanding feature, i.e., the exploration and exploitation of constraint interdependencies during ConflictAnalysis. This can improve propagation between constraints and contribute to the advanced selection heuristics. In contrast to our translation-based approach, CDNL-LNG requires scheduling of external propagation, an area that may require further investigation, but can improve efficiency.

An empirical analysis with our prototypical implementation has shown significant computational impact over translation-based and hybrid CASP systems, in particular when an eager encoding is large and ConflictAnalysis can discover useful constraint interdependencies. Some benchmarks also advocate a dedicated treatment of infeasible encodings via external propagation. Given the experimental evidence provided in Ohrimenko et al. (2009); Schutt et al. (2009), we also expect computational advantage over the state-of-the-art in CP systems, but leave further empirical study to future work.

Another open question is when to generate lazy nogoods. Recall that external propagators generate their inference whenever pruning is done. An interesting alternative might be to enforce the result of a propagator without requiring the unit nogood as evidence. Instead, each propagator maintains a propagation stack that stores enough data to reconstruct its state whenever the nogood is explicitly requested by the ASP solver, e.g., within ConflictAnalysis. There is some promising experimental evidence for the viability of this approach (Gent et al., 2010).

Left open to future work is also a study on exploiting LNG for implementing staging mechanisms, e.g., the lazy decomposition of an external propagator into simpler ones. This may lead to further computational impact.

Finally, given that eager and lazy encodings can be used interchangeably and

selected for each CP construct in the CASP encoding anew, an open question is how to automate this selection. A starting point for developing selection criteria might be a constraint's activity during the solving process (cf. Abío et al., 2013).

Chapter 7

Conclusion

This thesis contributed two alternative approaches to constraint answer set solving. For one, we introduced a translation-based approach, i.e., decomposing CASP into ASP. We laid detailed theoretical foundations in Chapter 4, where we showed how the problem of computing the constraint answer sets of a constraint program can then be reduced to computing the answer sets of a single, joint ASP encoding that includes an ASP representation of each finite domain variable and constraint that occurs in the original program. The reduction is facilitated by exploiting the splitting set theorem (Lifschitz and Turner, 1994). To our knowledge, this is the theorem's first significant application. Then, we presented various generic encodings of constraints based on the value, bound, range, or mixed representation of the variables' domains, and investigated the level of local consistency ASP inference can maintain. In Chapter 5, we proceeded with specialised encodings for some important global constraints, including ALL-DIFFERENT, GRAMMAR and related constraints, and REACHABILITY. Significant in our translation-based approach to constraint answer set solving are the following:

- Our ASP encodings are straightforward and easy to maintain, avoiding the need to integrating with additional specialised algorithms for a new constraint.

- By translation into ASP, the state of all constraint propagators is made transparent via (assignments to) atoms. Any ASP solver can use these in its advanced search heuristics, i.e., without need for programming search. Moreover, no scheduling is required as all constraints are always propagated at the same time.

- As our ASP encodings are formulated as preprocessing, any existing ASP systems can be made a CASP system without changing their source code. This allows for programmers to select the solver that best fit their needs.

- In particular, our approach seamlessly integrates with CDNL. Its CONFLICT-ANALYSIS can exploit constraint interdependencies. This can improve propagation between constraints and further contribute to search heuristics.

- We showed that in many cases, our ASP encodings achieve a level of local consistency equal to specialised algorithms, with a similar asymptotic run time complexity.

- Because of the flexibility of our approach and the advanced search methods available to us, however, we can also relax the encodings which makes our encodings even smaller and easier to maintain.

To our knowledge, previous work on modelling with ASP was not concerned with the propagation strength of encodings. In this regard, we made a first significant contribution to the field of ASP. In particular, the effect of unfounded set inference on the folklore ASP encoding of REACHABILITY was not understood. We here showed that the combination of UP, FL, BL, and LD inference maintains domain consistency of REACHABILITY. Whilst efficient implementations of UP and FL exist, the best known technique to compute WFJ was a combination of failed-literal-detection and FL (cf. Gebser and Schaub, 2013), having quadratic costs in size of the program. In Chapter 3, we devised a method that approximates the consequences of WFJ based on support flowgraphs, our novel graph-representation of programs written in ASP. We showed that the problem of finding all dominators in such graph can be used to approximate WFJ. Our techniques gave rise to new forms of unfounded set inference, called WFD and LD, which can lead to additional pruning. The results contributed in this thesis are significant for the following reasons:

- In contrast to the intuition that the inference in existing ASP systems, i.e., WFN and UP, naturally handles REACHABILITY, we showed that even some restricted variants of REACHABILITY cannot be efficiently propagated by a combination of UP and WFN.

- This gap can be closed with BL and LD, establishing practical relevance for these forms of inference. Yet, because of their quadratic run time complexity, BL and LD are currently not implemented by any existing ASP solver.

- The effects of WFJ and WFD, and BL and LD in particular, can be approximated in linear time, based on our reduction to the problem of finding all dominators in a flowgraph.

- Our approximation can be used to simulate the effects of BL and LD if the underlying program is component-unary. It even simulates the effects of WFJ and WFD if the program is unary. An important member from the class of component-unary programs is REACHABILITY. In general, any program can become (component-) unary as truth values are assigned during search.

- For above classes of programs, our method improves on the run time complexity of the best known method for computing consequences from BL and WFJ by a linear factor.

- Our techniques can be used to maintain domain consistency of REACHABILITY conditions.

This advocates further research into implementing propagators for unfounded sets, as they can provide highly useful extensions to ASP solving.

The other alternative approach to constraint answer set solving contributed in this thesis is centred around the integration of LNG with ASP solving. As detailed in Chapter 6, the idea is to incorporate external propagators to represent parts of the encoding implicitly, rather than generating it a-priori. However, external propagators can make an encoding of their inference explicit on demand. We laid their theoretical foundations based on variants of Lifschitz and Turner's splitting set theorem, and devised notions to characterise their propagation strength. We then developed CDNL-LNG, our conflict-driven algorithm for the problem of computing a constraint answer set. Beyond a CDNL-like decision procedure, it applies LNG via external propagation. The results contributed in this thesis are significant for the following reasons:

- LNG allows for representing encodings that are otherwise impractical. For instance, the best known ASP encoding of ALL-DIFFERENT that achieves domain consistency exploits a worst-case exponential number of Hall sets.

However, efficient domain consistency propagators for ALL-DIFFERENT exist (Régin, 1994). Constraint propagators are good candidates for efficient external propagators because they are well-studied.

– LNG integrates seamlessly with CDNL, i.e., the state-of-the-art in ASP solving. This includes nogood learning and backjumping.

– CDNL-LNG combines the key advantages of hybrid and translation-based approaches to CASP solving, that are, the use of efficient (external) propagators and the integration with CONFLICTANALYSIS.

– The techniques from LNG can be carried over to other conditions that are frequently expressed in ASP. In fact, LNG generalises existing approaches for the lazy encoding of consequences from FL (Gebser et al., 2007a) and weight constraints (Gebser et al., 2009b).

– It is easy to build notions for characterising the propagation strength of external propagators. For instance, Gebser et al.'s implementation of FL inference can be viewed as conflict-optimal external propagation of a program's loop nogoods, i.e., pruning more values than the approaches taken by Lin and Zhao (2002) and Giunchiglia et al. (2006). Though, Gebser et al.'s propagator is not inference-optimal.

The outstanding key advantage of both our approaches to constraint answer set solving is that they integrate seamlessly with CONFLICTANALYSIS. This allows for the exploitation constraint interdependencies, improving propagation between constraints and contributing to search heuristics. It also distinguishes our work from hybrid approaches to CASP solving which delegate the tasks of handling finite domain variables and the propagation of constraints to CP systems, and apply indirect measures to learn from the interdependencies between constraints.

– The success of our methodology is demonstrated by our prototypical CASP system *inca*. In 2013, it has successfully participated in a competition, finishing as first runner-up and outperforming hybrid CASP systems.

The challenge inherent with our techniques, however, is to define a compact representation of each constraint anew, a task that is more difficult in the translation-based approach without sacrificing propagation strength. In particular, the latter may not always scale due to an increase in space. Given this, hybrid CASP solving

is also a viable approach. In fact, we have to accept the idea that a mix of techniques is likely better than any single approach. The holy grail ought to be a CASP system which allows the programmer to choose a setting that best fits the problem domain.

Acronyms

ASP answer set programming

BL backward loop

CASP constraint answer set programming

CDNL conflict-driven nogood learning

CFG context-free grammar

CP constraint programming

CSP constraint satisfaction problem

CYK Cocke-Younger-Kasami

DFA deterministic finite automaton

FL forward loop

LD loop domination

LNG lazy nogood generation

SAT Boolean satisfiability

SCC strongly connected component

SMT satisfiability modulo theories

UP unit propagation

WFD well-founded domination

WFJ well-founded justification

WFN well-founded negation

Bibliography

A. Aavani, X. Wu, S. Tasharrofi, E. Ternovska, and D. G. Mitchell. Enfragmo: A system for modelling and solving search problems with logic. In *Proceedings of LPAR'12*, pages 15–22. Springer, 2012.

I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and P. J. Stuckey. To encode or to propagate? The best choice for each constraint in SAT. In *Proceedings of CP'13*, pages 97–106. Springer, 2013.

M. Alviano, F. Calimeri, G. Charwat, M. Dao-Tran, C. Dodaro, G. Ianni, T. Krennwallner, M. Kronegger, J. Oetsch, and A. Pfandler. The fourth answer set programming competition: Preliminary report. In *Proceedings of LPNMR'13*, pages 42–53. Springer, 2013a.

M. Alviano, C. Dodaro, W. Faber, N. Leone, and F. Ricca. WASP: A native ASP solver based on constraint learning. In *Proceedings of LPNMR'13*, pages 54–66. Springer, 2013b.

C. Anger, M. Gebser, and T. Schaub. Approaching the core of unfounded sets. In *Proceedings of NMR'06*, pages 58–66, 2006.

K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1988.

R. A. Aziz, P. J. Stuckey, and Z. Somogyi. Inductive definitions in constraint programming. In *Proceedings of ACSC'13*, pages 41–50. Australian Computer Society, 2013.

F. Bacchus. GAC via unit propagation. In *Procedings of CP'07*, pages 133–147. Springer, 2007.

M. Balduccini. Representing constraint satisfaction problems in answer set programming. In *ICLP Workshop ASPOCP'09*, 2009.

M. Balduccini. Industrial-size scheduling with ASP+CP. In *Proceedings of LPNMR'11*, pages 284–296. Springer, 2011.

M. Balduccini. ASP with non-Herbrand partial functions: a language and system for practical use. *Theory and Practice of Logic Programming*, 13(4-5):547–561, 2013.

M. Balduccini and Y. Lierler. Practical and methodological aspects of the use of cutting-edge asp tools. In *Proceedings of PADL'12*, pages 78–92. Springer, 2012.

M. Balduccini and Y. Lierler. Hybrid automated reasoning tools: from black-box to clear-box integration. In *ICLP Workshop ASPOCP'13*, 2013.

M. Banbara, M. Gebser, K. Inoue, T. Schaub, T. Soh, N. Tamura, and M. Weise. Aspartame: Solving constraint satisfaction problems with answer set programming. In *ICLP Workshop ASPOCP'13*, 2013.

C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

S. Baselice, P. Bonatti, and M. Gelfond. Towards an integration of answer set and constraint solving. In *Proceedings of ICLP'05*, pages 52–66. Springer, 2005.

R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of AAAI'97*, pages 203–208. AAAI Press/The MIT Press, 1997.

G. Benedek. *Hidato: 200 Pure Logic Puzzles*. Andrews McMeel Publishing, 2008.

C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in SAT. In *Proceedings of SAT'03*, pages 299–314. Springer, 2003.

C. Bessière, G. Katsirelos, N. Narodytska, C.-G. Quimper, and T. Walsh. Decompositions of all different, global cardinality and related constraints. In *Proceedings of IJCAI'09*, pages 419–424. AAAI Press/The MIT Press, 2009a.

C. Bessière, G. Katsirelos, N. Narodytska, and T. Walsh. Circuit complexity and decompositions of global constraints. In *Proceedings of IJCAI'09*, pages 412–418, 2009b.

C. Bessière, E. Hebrard, G. Katsirelos, and T. Walsh. Reasoning about connectivity constraints. In *Proceedings of IJCAI'15*, page To appear. IJCAI/AAAI Press, 2015.

A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009.

J. Bomanson and T. Janhunen. Normalizing cardinality rules using merging and sorting constructions. In *Proceedings of LPNMR'13*, pages 187–199. Springer, 2013.

G. Brewka, I. Niemelä, and M. Truszczyński. Nonmonotonic reasoning. In V. Lifschitz, F. van Harmelen, and B. Porter, editors, *Handbook of Knowledge Representation*, pages 239–284. Elsevier, 2008.

G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

F. Calimeri, W. Faber, N. Leone, and G. Pfeifer. Pruning operators for answer set programming systems. In *Proceedings of NMR'02*, pages 200–209, 2002.

F. Calimeri, G. Ianni, F. Ricca, M. Alviano, A. Bria, G. Catalano, S. Cozza, W. Faber, O. Febbraro, N. Leone, M. Manna, A. Martello, C. Panetta, S. Perri, K. Reale, M. C. Santoro, M. Sirianni, G. Terracina, and P. Veltri. The third answer set programming competition: Preliminary report of the system competition track. In *Proceedings of LPNMR'11*, pages 388–403. Springer, 2011.

M. Celik, H. Erdogan, F. Tahaoglu, T. Uras, and E. Erdem. Comparing ASP and CP on four grid puzzles. In *AIIA Workshop RCRA'09*, 2009.

P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of IJCAI'91*, pages 331–340. Morgan Kaufmann, 1991.

X. Chen, J. Ji, and F. Lin. Computing loops with at most one external support rule. *ACM Transactions on Computational Logic*, 14(1):3:1–3:34, 2013.

N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

K. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

E. Coban, E. Erdem, and F. Türe. Comparing ASP, CP, ILP on two challenging applications: Wire routing and haplotype inference. In *Proceedings of LaSh'08*, 2008.

M.-C. Côté, B. Gendron, C.-G. Quimper, and L.-M. Rousseau. Formal languages for integer programming modeling of shift scheduling problems. *Constraints*, 16:54–76, 2011.

A. Dal Palù, A. Dovier, E. Pontelli, and G. Rossi. Answer set programming with constraints using lazy grounding. In *Proceedings of ICLP'09*, pages 115–129. Springer, 2009.

B. de Cat, M. Denecker, M. Bruynooghe, and P. J. Stuckey. Lazy model expansion: Interleaving grounding with search. *Journal of Artificial Intelligence Research*, 52:235–286, 2015.

R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

M. Denecker, J. Vennekens, S. Bond, M. Gebser, and M. Truszczyński. The second answer set programming competition. In *Proceedings of LPNMR'09*, pages 637–654, 2009.

M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of FGCS'88*, pages 693–702, 1988.

G. Dooms, Y. Deville, and P. Dupont. CP(Graph): Introducing a graph computation domain in constraint programming. In *Proceedings of CP'05*, pages 211–225. Springer, 2005.

A. Dovier, A. Formisano, and E. Pontelli. A comparison of CLP(FD) and ASP solutions to NP-complete problems. In *Proceedings of ICLP'05*, pages 67–82. Springer, 2005.

W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Logic Programming*, 1(3):267–284, 1984.

C. Drescher. Symmetry breaking for answer set programming. Master's thesis, Technische Universität Wien, 2010.

N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

T. Eiter, G. Brewka, M. Dao-Tran, M. Fink, G. Ianni, and T. Krennwallner. Combining nonmonotonic knowledge bases with external sources. In *Proceedings of FroCoS'09*, pages 18–42. Springer, 2009.

T. Eiter, M. Fink, T. Krennwallner, and C. Redl. Conflict-driven ASP solving with external sources. *Theory and Practice of Logic Programming*, 12(4-5):659–679, 2012.

I. Elkabani, E. Pontelli, and T. Son. Smodels with CLP and its applications: A simple and effective approach to aggregates in ASP. In *Proceedings of ICLP'04*, pages 73–89. Springer, 2004.

E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3(4-5):499–518, 2003.

W. Faber, G. Pfeifer, and N. Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

F. Fages. Consistency of Clark's completion and the existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

M. Fujita, J. K. Slaney, and F. Bennett. Automatic generation of some results in finite algebra. In *Proceedings of IJCAI'93*, pages 52–59. Morgan Kaufmann Publishers, 1993.

M. Gebser. *Proof theory and algorithms for answer set programming*. PhD thesis, Universität Potsdam, 2011.

M. Gebser and T. Schaub. Tableau calculi for logic programs under answer set semantics. *ACM Transactions on Computational Logic*, 14(2):15:1–15:40, 2013.

M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proceedings of IJCAI'07*, pages 386–392. AAAI Press/MIT Press, 2007a.

M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In *Proceedings of LPNMR'07*, pages 260–265. Springer, 2007b.

M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set enumeration. In *Proceedings of LPNMR'07*, pages 136–148. Springer, 2007c.

M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In *Proceedings of ICLP'08*, pages 190–205. Springer, 2008.

M. Gebser, H. Hinrichs, T. Schaub, and S. Thiele. xpanda: A (simple) preprocessor for adding multi-valued propositions to ASP. In *Proceedings of WLP'09*, 2009a.

M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. On the implementation of weight constraint rules in conflict-driven ASP solvers. In *Proceedings of ICLP'09*, pages 250–264. Springer, 2009b.

M. Gebser, M. Ostrowski, and T. Schaub. Constraint answer set solving. In *Proceedings of ICLP'09*, pages 235–249. Springer, 2009c.

M. Gebser, T. Grote, R. Kaminski, and T. Schaub. Reactive answer set programming. In *Proceedings of LPNMR'11*, pages 54–66. Springer, 2011a.

M. Gebser, R. Kaminski, A. König, and T. Schaub. Advances in *gringo* series 3. In *Proceedings of LPNMR'11*, pages 345–351. Springer, 2011b.

M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. T. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011c.

M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, and T. Schaub. Stream reasoning with answer set programming: Preliminary report. In *Proceedings of KR'12*, pages 613–617. AAAI Press, 2012a.

M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012b.

M. Gebser, H. Jost, R. Kaminski, P. Obermeier, O. Sabuncu, T. Schaub, and M. Schneider. Ricochet robots: A transverse ASP benchmark. In *Proceedings of LPNMR'13*, pages 348–360. Springer, 2013a.

M. Gebser, B. Kaufmann, R. Otero, J. Romero, T. Schaub, and P. Wanko. Domain-specific heuristics in answer set programming. In *Proceedings of AAAI'13*, pages 350–356. AAAI Press, 2013b.

M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP'88*, pages 1070–1080. The MIT Press, 1988.

I. P. Gent. The fundamental conjecture of reformulation. In *CP Workshop Formul'01*, 2001.

I. P. Gent. Arc consistency in SAT. In *Proceedings of ECAI'02*, pages 121–125. IOS Press, 2002.

I. P. Gent. Optimal implementation of watched literals and more general techniques. *Journal of Artificial Intelligence Research*, 48:231–251, 2013.

I. P. Gent and T. Walsh. CSPLIB: A benchmark library for constraints. In *Proceedings of CP'99*, pages 480–481. Springer, 1999.

I. P. Gent, I. Miguel, and P. Nightingale. Generalised arc consistency for the all-different constraint: An empirical survey. *Artificial Intelligence*, 172(18):1973–2000, 2008.

I. P. Gent, I. Miguel, and N. C. A. Moore. Lazy explanations for constraint propagators. In *Proceedings of PADL'10*, pages 217–233. Springer, 2010.

L. Georgiadis and R. E. Tarjan. Finding dominators revisited. In *Proceedings of SODA'04*, pages 869–878. SIAM, 2004.

E. Giunchiglia, Y. Lierler, and M. Maratea. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, 36(4):345–377, 2006.

E. I. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Proceedings of DATE'02*, pages 142–149. IEEE Computer Society, 2002.

C. P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proceedings of AAAI'97*, pages 221–226. AAAI Press/The MIT Press, 1997.

P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935.

J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

T. Janhunen. Comparing the expressive powers of some syntactically restricted classes of logic programs. In *Proceedings of CL'00*, pages 852–866. Springer, 2000.

T. Janhunen. Representing normal programs with clauses. In *Proceedings of ECAI'04*, pages 358–362. IOS Press, 2004.

T. Janhunen. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics*, 16(1-2):35–86, 2006.

T. Janhunen and I. Niemelä. Compact translations of non-disjunctive answer set programs to propositional clauses. In *Proceedings of Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, pages 111–130. Springer, 2011.

T. Janhunen, E. Oikarinen, H. Tompits, and S. Woltran. Modularity aspects of disjunctive stable models. *Journal of Artificial Intelligence Research*, 35:813–857, 2009.

M. Järvisalo, E. Oikarinen, T. Janhunen, and I. Niemelä. A module-based framework for multi-language constraint modeling. In *Proceedings of LPNMR'09*, pages 155–169. Springer, 2009.

G. Katsirelos, S. Maneth, N. Narodytska, and T. Walsh. Restricted global grammar constraints. In *Proceedings of CP'09*, pages 501–508. Springer, 2009a.

G. Katsirelos, N. Narodytska, and T. Walsh. Reformulating global grammar constraints. In *Proceedings of CPAIOR'09*, pages 132–147. Springer, 2009b.

M. Leconte. A bounds-based reduction scheme for constraints of difference. In *CP Workshop CONSTRAINT'95*, 1996.

J. Lee. A model-theoretic counterpart of loop formulas. In *Proceedings of IJCAI'05*, pages 503–508. Professional Book Center, 2005.

C. Lefèvre and P. Nicolas. The first version of a new ASP solver : ASPeRiX. In *Proceedings of LPNMR'09*, pages 522–527. Springer, 2009.

N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

Y. Lierler. Abstract answer set solvers with backjumping and learning. *Theory and Practice of Logic Programming*, 11(2-3):135–169, 2011.

Y. Lierler. Relating constraint answer set programming languages and algorithms. *Artificial Intelligence*, 207:1–22, 2014.

Y. Lierler and M. Truszczynski. Abstract modular inference systems and solvers. In *Proceedings of PADL'14*, pages 49–64. Springer, 2014.

Y. Lierler, S. Smith, M. Truszczynski, and A. Westlund. Weighted-sequence problem: ASP vs CASP and declarative vs problem-oriented solving. In *Proceedings of PADL'12*, pages 63–77. Springer, 2012.

V. Lifschitz. Answer set planning. In *Proceedings of ICLP'99*, pages 23–37, 1999.

V. Lifschitz. What is answer set programming? In *Proceedings of AAAI'08*, pages 1594–1597. AAAI Press, 2008a.

V. Lifschitz. Twelve definitions of a stable model. In *Proceedings of ICLP'08*, pages 37–51. Springer, 2008b.

V. Lifschitz and A. Razborov. Why are there so many loop formulas? *ACM Transactions on Computational Logic*, 7(2):261–268, 2006.

V. Lifschitz and H. Turner. Splitting a logic program. In *Proceedings of ICLP'94*, pages 23–37, 1994.

F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of AAAI'02*, pages 112–118. AAAI Press/MIT Press, 2002.

G. Liu, T. Janhunen, and I. Niemelä. Answer set programming via mixed integer programming. In *Proceedings of KR'12*, pages 32–42. AAAI Press, 2012.

T. Mancini, D. Micaletto, F. Patrizi, and M. Cadoli. Evaluating ASP and commercial solvers on the CSPLib. *Constraints*, 13(4):407–436, 2008.

M. Maratea, F. Ricca, W. Faber, and N. Leone. Look-back techniques and heuristics in DLV: Implementation, evaluation, and comparison to QBF solvers. *Journal of Algorithms*, 63(1-3):70–89, 2008.

V. M. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K.R. Apt, V.W. Marek, M. Truszczyński, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25-year perspective*, pages 375–398. Springer, 1999.

J. P. Marques-Silva and K A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

V. Mellarkod and M. Gelfond. Integrating answer set reasoning with constraint solving techniques. In *Proceedings of FLOPS'08*, pages 15–31. Springer, 2008.

V. Mellarkod, M. Gelfond, and Y. Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287, 2008.

D. Mitchell. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science*, 85:112–133, 2005.

U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of DAC'01*, pages 530–535. ACM, 2001.

I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

I. Niemelä. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):313–329, 2008.

I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proceedings of NMR'99*, pages 317–333. Springer, 1999.

R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

O. Ohrimenko, P. J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

E. Oikarinen and T. Janhunen. Modular equivalence for normal logic programs. In *Proceedings of ECAI'06*, pages 412–416. IOS Press, 2006.

M. Ostrowski and T. Schaub. ASP modulo CSP: The clingcon system. *Theory and Practice of Logic Programming*, 12(4-5):485–503, 2012.

N. Pelov. *Semantics of logic programs with aggregates*. PhD thesis, Department of Computer Science, K.U. Leuven, Belgium, April 2004.

G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proceedings of CP'04*, pages 482–495. Springer, 2004.

C.-G. Quimper and T. Walsh. Global grammar constraints. In *Proceedings of CP'06*, pages 751–755. Springer, 2006.

C.-G. Quimper and T. Walsh. Decomposing global grammar constraints. In *Proceedings of CP'07*, pages 590–604. Springer, 2007.

C.-G. Quimper and T. Walsh. Decompositions of grammar constraints. In *Proceedings of AAAI'08*, pages 1567–1570. AAAI Press, 2008.

J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI'94*, pages 362–367, 1994.

F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

A. Schutt, T. Feydy, P. J. Stuckey, and M. Wallace. Why cumulative decomposition is not as bad as it sounds. In *Proceedings of CP'09*, pages 746–761. Springer, 2009.

M. Sellmann. The theory of grammar constraints. In *Proceedings of CP'06*, pages 530–544. Springer, 2006.

P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proceedings of CP'05*, pages 827–831. Springer, 2005.

V. C. Sreedhar, G. R. Gao, and Y.-F. Lee. Incremental computation of dominator trees. *ACM Transactions on Programming Languages and Systems*, 19(2):239–252, 1997.

N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. In *Proceedings of CP'06*, pages 590–603. Springer, 2006.

R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

P. Van Hentenryck, V. A. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(FD). *Logic Programming*, 37(1-3):139–164, 1995.

W. J. van Hoeve. The alldifferent constraint: A survey. *Computing Research Repository*, cs.PL/0105015, 2001.

T. Walsh. SAT v CSP. In *Proceedings of CP'00*, pages 441–456. Springer, 2000.

S. Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

J. Wittocx, M. Mariën, and M. Denecker. The IDP system: A model expansion system for an extension of classical logic. In *Proceedings of LASH'08*, pages 153–165. ACCO, 2008.

J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of DIALM '99*, pages 7–14. ACM Press, 1999.

J.-H. You and G. Hou. Arc-consistency + unit propagation = lookahead. In *Proceedings of ICLP'04*, pages 314–328. Springer, 2004.

D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):372–375, 1967.

L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings DATE'03*, pages 10880–10885. IEEE Computer Society, 2003.

L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in Boolean satisfiability solver. In *Proceedings of ICCAD'01*, pages 279–285, 2001.

N.-F. Zhou. The language features and architecture of B-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):189–218, 2012.