Modelling Grammar Constraints with Answer Set Programming

Christian Drescher and Toby Walsh

NICTA and University of New South Wales
Locked Bag 6016, Sydney NSW 1466, Australia
E-mail: {Christian.Drescher,Toby.Walsh}@nicta.com.au

Abstract

Representing and solving constraint satisfaction problems is one of the challenges of artificial intelligence. In this paper, we present answer set programming (ASP) models for an important and very general class of constraints, including all constraints specified via grammars or automata that recognise some formal language. We argue that our techniques are effective and efficient, e.g., unit-propagation of an ASP solver can achieve domain consistency on the original constraint. Experiments demonstrate computational impact.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases answer set programming, grammar-, regular-, precedence constraint

Digital Object Identifier 10.4230/LIPIcs.ICLP.2011.28

1 Introduction

Answer set programming (ASP; [3]) provides a compact, declarative, and highly competitive approach to modelling and solving constraint satisfaction problems (CSP) [21, 9, 11]. CSP are combinatorial problems defined as a set of variables whose value must satisfy a number of limitations (the constraints), and stem from a variety of areas. One very promising method for scheduling, rostering and sequencing problems is to specify constraints via grammars or automata that recognise some formal language [23, 27, 25, 16]. 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*. Grammar-based constraint propagators were proposed in [27, 25], and modelled with Boolean satisfiability (SAT; [4]) in [26, 1], while an automata-based constraint propagator was presented in [23], and modelled with SAT in [2]. Whilst SAT models can be directly converted into ASP [21], 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. First, we show that the GRAMMAR constraint can be modelled with ASP based on the production rules in the grammar. Second, we present alternative ASP models for a special case of the GRAMMAR constraint, that is, the REGULAR constraint, based on deterministic finite automata. Third, we give theoretical results on the pruning achieved by unit-propagation of an ASP solver, and runtime complexity. Forth, we provide an ASP model for the PRECEDENCE constraint [18] which is a special case of the REGULAR constraint. It is useful for breaking value symmetry in CSP. We argue that our encoding improves runtime complexity over REGULAR. Finally, we demonstrate the applicability of our approach on *shift-scheduling* and *graph colouring* instances. We show that symmetry breaking using our ASP models yield significant improvements in runtime, and outperforms recent generic approaches to symmetry breaking for ASP.

Technical Communications of the 27th International Conference on Logic Programming (ICLP'11).
Editors: John P. Gallagher, Michael Gelfond; pp. 28–39

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Background

Answer Set Programming

A *(normal) logic program* over a set of primitive propositions $\mathcal{A}$, $\bot \in \mathcal{A}$, is a finite set of *rules* r of the form

$$a_0 \leftarrow a_1, .., a_m, not\ a_{m+1}, .., not\ a_n$$

where $a_i \in \mathcal{A}$ are *atoms* for $0 \leq i \leq n$, and $\{a_1, .., a_m, not\ a_{m+1}, .., not\ a_n\}$ is the *body* of r. We also consider *choice rules* of the form

$$\{h_1, .., h_k\} \leftarrow a_1, .., a_m, not\ a_{m+1}, .., not\ a_n$$

that allow for the nondeterministic choice over atoms in $\{h_1, .., h_k\}$. Their semantics is given through program transformations [28]. The answer sets of a logic program Π can be characterised as Boolean assignments A over the atoms and bodies in Π, $dom(A)$, that satisfy nogoods imposed by Π [13]. Formally, an *assignment* A is a set $\{\sigma_1, .., \sigma_n\}$ of *(signed) literals* σ_i expressing that $a \in dom(A)$ is assigned *true* or *false*. The proposition $\bot$ denotes an atom that is false in every assignment. In our context, a *nogood* [8] is a set $\delta = \{\sigma_1, .., \sigma_m\}$ of literals, expressing a condition *violated* by any assignment A if $\delta \subseteq A$. Nogoods allow for a transparent technology transfer from SAT since every nogood can be syntactically represented by a clause, for instance, inferences in ASP can be viewed as unit-propagation on nogoods [13]. For a nogood δ, a literal $\sigma \in \delta$, and an assignment A, we say that δ is *unit* and the complement of σ is *unit-resulting* if $\delta \setminus A = \{\sigma\}$. *Unit-propagation* is the process of repeatedly extending A by unit-resulting literals until no unit nogood remains or is violated. A total assignment A is a *solution* to a set of nogoods Δ if $\delta \not\subseteq A$ for all $\delta \in \Delta$. Given a logic program Π, one can specify a set of nogoods such that its solutions capture the models of the Clark's completion of Π [6, 13]. We denote this set $\Delta(\Pi)$. If Π is *tight* [12], the solutions to $\Delta(\Pi)$ are precisely the answer sets of Π. (All logic programs presented in this paper are tight.) Otherwise, loop formulas, expressed in the set of nogoods $\Lambda(\Pi)$, have to be added to establish full correspondence to the answer sets of Π [19].

Constraint Satisfaction

We want to use ASP to model and solve CSP. Formally, a CSP is a triple (V, D, C) where V is a finite set of *variables*, D is a set of finite *domains* such that each variable $\mathsf{v} \in V$ has an associated domain $dom(\mathsf{v}) \in D$, and C is a set of constraints. A *constraint* c is a pair (R_S, S) where R_S is an n-ary *relation* on the variables in $S \in V^n$, called the *scope* of c. A *(variable) assignment* $\mathsf{v} \mapsto t$ means that variable $\mathsf{v} \in V$ is mapped to the value $t \in dom(\mathsf{v})$. A *(compound) assignment* $(\mathsf{v}_1, .., \mathsf{v}_m) \mapsto (t_1, .., t_m)$ denotes the assignments $\mathsf{v}_i \mapsto t_i$, $1 \leq i \leq m$. A constraint $c = (R_S, S)$ is *satisfied* iff $S \mapsto (t_1, .., t_m)$ and $(t_1, .., t_m) \in R_S$. A *solution* of a CSP is an assignment for all variables in V satisfying all constraints in C. Constraint solvers typically use backtracking search to explore assignments in a search tree. In a search tree, each node represents an assignment to some variables, child nodes are obtained by selecting an unassigned variable and having a child node for each possible value for this variable, and the root node is empty. Every time a variable is assigned a value, a propagation stage is executed, pruning the set of values for the other variables, i.e., enforcing a certain type of local consistency such as domain consistency. An n-ary constraint $c = (R_S, S)$ is *domain consistent* iff whenever a variable $\mathsf{v}_i \in S$ is assigned a value $t_i \in dom(\mathsf{v}_i)$, there exist compatible values in the domains of all the other variables $t_j \in dom(\mathsf{v}_j)$ for all $1 \leq j \leq n,\ j \neq i$ such that $(t_1, .., t_n) \in R_S$, forming a *support* for $\mathsf{v}_i \mapsto t_i$.

Grammar Constraints

We will consider constraints requiring that values taken by the sequence of variable in its scope belong to a formal language generated by a context-free grammar or accepted by an automaton. A *context-free grammar* (CFG; [5]) is formally defined as a quadruple $\mathcal{G} = (N, \Sigma, P, S)$, where N is a finite set of *nonterminal* symbols, Σ is a finite set of *terminal* symbols (disjoint from N), $P \subseteq N \times (N \cup \Sigma)^*$ is a set of *production* rules, and the distinguished *start nonterminal* $S \in N$. We often omit to specify the complete quadruple and only provide the set of productions using the following conventions: capital letters denote nonterminals in N, lowercase letters denote terminals in Σ, and υ and ω denote a sequence of letters called *string*. We also assume that S is the start nonterminal. Moreover, productions $(A, \omega) \in P$ can be written as $A ::= \omega$, and productions $(A, \omega_1), .., (A, \omega_m) \in P$ can be written as $A ::= \omega_1 \mid .. \mid \omega_m$. We write $\upsilon_1 A \upsilon_2 \Rightarrow_{\mathcal{G}} \upsilon_1 \omega \upsilon_2$ iff $(A, \omega) \in R$, $\omega_1 \Rightarrow^*_{\mathcal{G}} \omega_m$ iff there exists a sequence of strings $\omega_2, .., \omega_{m-1}$ such that $\omega_i \Rightarrow_{\mathcal{G}} \omega_{i+1}$ for all $1 \leq i < m$. For $\omega_1 \Rightarrow^*_{\mathcal{G}} \omega_m$ we say that ω_1 *generates* ω_m and ω_1 is *derived from* ω_m. The *language produced by* $\mathcal{G}$ is the set of strings $L_{\mathcal{G}} = \{\omega \in \Sigma^* \mid S \Rightarrow^*_{\mathcal{G}} \omega\}$. A CFG is in *Chomsky normal form* iff all productions are of the form $A ::= a$ or $A ::= BC$. Every CFG $\mathcal{G}$ such that the empty string ε is not generated by $\mathcal{G}$ can be transformed into a grammar $\mathcal{H}$ such that $L_{\mathcal{G}} = L_{\mathcal{H}}$ and $\mathcal{H}$ is in Chomsky normal form. Transformations are described in most textbooks on automata theory, such as [14], with at most a linear increase in the size of the grammar. Given a CFG $\mathcal{G}$, the GRAMMAR constraint GRAMMAR$(\mathcal{G}, [\mathsf{v}_1, .., \mathsf{v}_n])$ is satisfied by just those assignments to the sequence of variables $(\mathsf{v}_1, .., \mathsf{v}_n)$ which belong to the language produced by $\mathcal{G}$ [27, 25].

The REGULAR constraint [23] REGULAR$(\mathcal{G}, [\mathsf{v}_1, .., \mathsf{v}_n])$ is a special case of the GRAMMAR constraint, i.e., it accepts just those assignments to the sequence of variables $(\mathsf{v}_1, .., \mathsf{v}_n)$ which belong to the regular language, that is, produced by a regular grammar $\mathcal{G}$. As we are recognising only strings are of a fixed length, a transformation of grammar constraints into regular constraints which may increase the space required to represent the constraint is presented in [17]. Regular languages are strictly contained within context-free languages, and can be specified with productions of the form $A ::= t$ or $A ::= tB$. Alternatively, regular languages can be specified by means of an automaton. A *deterministic finite automaton* (DFA) 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 $Q \times \Sigma \to Q$, 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 M to perform a transition from its current state q to a new state $\delta(q, t)$, where M starts off in the state q_0. The input ω is *recognised* by M iff ω causes M to transition from q_0 in one of the accepting states. The *language recognised by* M is the set of inputs $L_M = \{\omega \in \Sigma^* \mid M \text{ recognises } \omega\}$. Given a DFA M, the REGULAR constraint REGULAR$(M, [\mathsf{v}_1, .., \mathsf{v}_n])$ is satisfied on just those assignments to the sequence of variables $(\mathsf{v}_1, .., \mathsf{v}_n)$ which belong to the language recognised by M.

The PRECEDENCE constraint [18] is a special case of the REGULAR constraint. It is used for breaking symmetries of interchangeable values in a CSP. A pair of values are *interchangeable* if we can swap them in any solution. Pairwise precedence of s over t in a sequence of variables $(\mathsf{v}_1, .., \mathsf{v}_n)$, denoted as PRECEDENCE$([s, t], [\mathsf{v}_1, .., \mathsf{v}_n])$, holds iff whenever there exists j such that $\mathsf{v}_j \mapsto t$, then there must exist $i < j$ such that $\mathsf{v}_i \mapsto s$. Many CSPs, however, involve multiple interchangeable values, not just two. For instance, when we assign colours to vertices in the *graph colouring* problem, all values are interchangeable. Then, the PRECEDENCE constraint PRECEDENCE$([t_1, .., t_m], [\mathsf{v}_1, .., \mathsf{v}_n])$ holds iff $\min(\{i \mid \mathsf{v}_i \mapsto t_k\} \cup \{n+1\}) < \min(\{i \mid \mathsf{v}_i \mapsto t_\ell\} \cup \{n+2\})$ for all $1 \leq k < \ell < m$. A similar idea for breaking symmetries in the *planning* domain is presented in [15].

Modelling Multi-valued Variables

We want to model GRAMMAR, REGULAR, and PRECEDENCE constraints with ASP. Their encoding requires access to information on the variables $\mathsf{v}_1, .., \mathsf{v}_n$ in the scope of the constraint, i.e., information on possible assignments $\mathsf{v}_i \mapsto t$ with $t \in dom(\mathsf{v})$. In this paper, we will follow previous work on modelling CSP with ASP [11], and maintain a consistent set of assignments through translation into ASP together with our constraint encodings. Each possible assignment $\mathsf{v}_i \mapsto t$ is represented by an atom $[\![\mathsf{v}_i \mapsto t]\!]$. It is false iff t has been pruned from the domain of v_i. Conversely, it is true iff $\mathsf{v}_i \mapsto t$. The following rules, collected in $encode([\mathsf{v}_1, .., \mathsf{v}_n])$, ensure that one and only one value $t_{i_j} \in \Sigma$ can be assigned, for $1 \leq i \leq n$ and $dom(\mathsf{v}_i) = \{t_{i_1}, .., t_{i_m}\}$.

$$\{[\![\mathsf{v}_i \mapsto t_{i_1}]\!], .., [\![\mathsf{v}_i \mapsto t_{i_m}]\!]\} \leftarrow$$
$$\bot \leftarrow [\![\mathsf{v}_i \mapsto t_{i_j}]\!], [\![\mathsf{v}_i \mapsto t_{i_k}]\!], j \neq k$$
$$\bot \leftarrow not\ [\![\mathsf{v}_i \mapsto t_{i_1}]\!], .., not\ [\![\mathsf{v}_i \mapsto t_{i_m}]\!]$$

Although there are $\mathcal{O}(nm^2)$ nogoods resulting from this ASP model, unit-propagation is performed on these nogoods in $\mathcal{O}(nm)$. A more compact $\mathcal{O}(nm)$ representation that propagates in $\mathcal{O}(nm)$ time is presented in [11, 28].

3 Modelling the Grammar Constraint

We show here how to model GRAMMAR$(\mathcal{G}, [\mathsf{v}_1, .., \mathsf{v}_n])$ with ASP in a very straightforward way, based on the well known CYK parser [29] which requires the CFG to be in Chomsky normal form. Our encoding of the CYK parser constructs a parsing table T where each $A \in T[i, j]$ is a nonterminal symbol that is derived from a substring ω of j symbols starting at the ith symbol such that the assignment $(\mathsf{v}_i, \mathsf{v}_{i+1}, .., \mathsf{v}_{i+j-1}) \mapsto \omega$ is possible. Our $\mathcal{O}(|P|n^3)$ sized ASP model, denoted as $encode(\mathcal{G})$, is as follows:

1. We introduce new atoms $A(i, j)$ which are true iff $A \in A[i, j]$. A consistent assignment to $A(i, j)$ is enforced by the rules that follow below.
2. Each production of the form $A ::= t$ is encoded by
   $$A(i, 1) \leftarrow [\![\mathsf{v}_i \mapsto t]\!]$$
   which states that A can be derived from the ith symbol if $\mathsf{v}_i \mapsto t$, i.e., the ith symbol is t.
3. Each production of the form $A ::= BC$ is encoded by
   $$A(i, j) \leftarrow B(i, k), C(i+k, j-k).$$
   Intuitively, above rule states that A is derived from the string starting at the ith symbol of length j if B is derived from the substring starting at the ith symbol of length k, and C is derived from the substring starting at the $i+k$th symbol of length $j-k$. In other words, k splits the string generated by A into the substrings B and C.
4. Finally, the condition that the start nonterminal S has to be derived from the input string is captured by
   $$\bot \leftarrow not\ S(1, n)$$
   which expresses that every answer set of the resulting ASP model contains $S(1, n)$.

▶ **Theorem 1.** *The assignments satisfying* GRAMMAR$(\mathcal{G}, [\mathsf{v}_1, .., \mathsf{v}_n])$ *correspond one-to-one to the answer sets of* $encode(\mathcal{G}) \cup encode([\mathsf{v}_1, .., \mathsf{v}_n])$.

The theorem follows from the proof of correctness of the CYK parser. Unfortunately, our straightforward ASP model is not very efficient from a theoretical point of view, i.e., it does not prune all possible values.

▶ **Example 2.** Consider the following CFG $\mathcal{G}$ given through the productions $S ::= SA \mid AS \mid b$ and $A ::= a$. Suppose the input sequence of length 2, $(\mathsf{v}_1, \mathsf{v}_2)$ with $dom(\mathsf{v}_1) = dom(\mathsf{v}_2) = \{a, b, c\}$. Our model $encode(\mathcal{G})$ comprise the following rules

$$A(1,1) \leftarrow [\![\mathsf{v}_1 \mapsto a]\!] \qquad S(1,1) \leftarrow [\![\mathsf{v}_1 \mapsto b]\!] \qquad S(1,2) \leftarrow S(1,1), A(2,1)$$
$$A(2,1) \leftarrow [\![\mathsf{v}_2 \mapsto a]\!] \qquad S(2,1) \leftarrow [\![\mathsf{v}_2 \mapsto b]\!] \qquad S(1,2) \leftarrow A(1,1), S(2,1)$$
$$\bot \leftarrow not\ S(1,2)$$

Although the value c does not appear in a satisfying assignment for $\textsc{grammar}(\mathcal{G}, [\mathsf{v}_1, \mathsf{v}_2])$, unit-propagation on $\Delta(encode(\mathcal{G}) \cup encode([\mathsf{v}_1, \mathsf{v}_2]))$ does not prune c from the domains.

A SAT model of the GRAMMAR constraint, based on and-or-graphs, such that unit-propagation achieves domain consistency on the original constraint was presented in [26]. To achieve a similar result, we revise our ASP model that we now denote $encode^{DC}(\mathcal{G})$. The idea is that $A(i,j)$ is set to true iff the nonterminal A participates in a successful parsing of the input string starting from the start nonterminal S.

1. We introduce atoms $A(i,j)$ which are true iff $S \Rightarrow^*_{\mathcal{G}} \omega_1 A \omega_2 \Rightarrow^*_{\mathcal{G}} \omega$ for some ω_1, ω_2, where ω is the input string and A is derived by the substring starting from i of length j. Similarly, we introduce atoms $\omega_{BC}(i,j)$ which indicate whether $S \Rightarrow^*_{\mathcal{G}} \omega_1 BC \omega_2 \Rightarrow^*_{\mathcal{G}} \omega$ and the two nonterminals BC is derived by the substring starting from i of length j. A consistent assignment to $A(i,j)$, $\omega_{BC}(i,j)$ respectively, is enforced by the rules that follow below.
2. Each production of the form $A ::= t$ is now encoded by
   $$\{A(i,1)\} \leftarrow [\![\mathsf{v}_i \mapsto t]\!] .$$
   To ensure that unit-propagation prunes all possible values, we capture the condition that for each assignment $\mathsf{v}_i \mapsto t$ there exist a nonterminal A that generates t at the ith symbol and participates in a successful parsing starting from S. Let $A_1, .., A_m$ be all nonterminals such that $A_\ell ::= t$ for $1 \leq \ell \leq m$. We encode *support* for $\mathsf{v}_i \mapsto t$ by
   $$\bot \leftarrow [\![\mathsf{v}_i \mapsto t]\!], not\ A_1(i,1), .., not\ A_m(i,1) .$$
3. Each production of the form $A ::= BC$ is encoded by
   $$\{A(i,j)\} \leftarrow \omega_{BC}(i,j) \qquad \{\omega_{BC}(i,j)\} \leftarrow B(i,k), C(i+k, j-k)$$
   stating that if a production for A applies (e.g., the string ω_{BC}), then A may or may not be in a parsing starting from S. To ensure that unit-propagation prunes all possible values, we have to encode the condition that whenever $\omega_{BC}(i,j)$ is true then there must be a nonterminal A that generates BC, i.e., $S \Rightarrow^*_{\mathcal{G}} \omega_1 A \omega_2 \Rightarrow_{\mathcal{G}} \omega_1 BC \omega_2 \Rightarrow^*_{\mathcal{G}} \omega$ for some ω_1, ω_2, where ω is the input string and $A ::= BC$. Let $A_1, .., A_m$ be all such nonterminal A. Then we encode this condition by
   $$\bot \leftarrow \omega_{BC}(i,j), not\ A_1(i,j), .., not\ A_m(i,j) .$$
   Similarly, we encode support for each nonterminal $A' \in N \setminus \{S\}$, i.e., A' must be in the right-hand-side of a production, say the string ω_{BC} for $A' = B$ or $A' = C$, such that $S \Rightarrow^*_{\mathcal{G}} \omega_1 \omega_{BC} \omega_2 \Rightarrow_{\mathcal{G}} \omega_1 BC \omega_2 \Rightarrow^*_{\mathcal{G}} \omega$ for some ω_1, ω_2, where ω is the input string. Let $\omega_{BC,1}, .., \omega_{BC,m}$ be all such strings ω_{BC}. Then we encode this condition by
   $$\bot \leftarrow A'(i,j), not\ \omega_{BC,1}(i_1, j_1), .., not\ \omega_{BC,m}(i_m, j_m) .$$
   Observe that for all $1 \leq \ell \leq m$ we have either $i_\ell = i$ or $j_\ell = j$. Hence, there are only $\mathcal{O}(|P|n^3)$ rules from this item, i.e., $encode^{DC}(\mathcal{G})$ does not increase the space complexity over $encode(\mathcal{G})$.
4. The condition that the starting nonterminal S has to generate the input string remains unchanged.
   $$\bot \leftarrow not\ S(1,n)$$

Altogether, the rules in $encode^{DC}(\mathcal{G})$ enforce that whenever $A(i,j)$ is in an answer set then the nonterminal A is used to generate the substring at the ith symbol of length j in a successful parsing starting from S. This observation also applies to $\omega_{BC}(i,j)$.

▶ **Example 3.** Consider again the CFG $\mathcal{G}$ from Example 2, again applied to $(\mathsf{v}_1, \mathsf{v}_2)$ with $dom(\mathsf{v}_1) = dom(\mathsf{v}_2) = \{a,b,c\}$. Our ASP model $encode^{DC}(\mathcal{G})$ comprises the following rules

$$\{A(1,1)\} \leftarrow [\![\mathsf{v}_1 \mapsto a]\!] \quad \{S(1,1)\} \leftarrow [\![\mathsf{v}_1 \mapsto b]\!] \quad \{\omega_{SA}(1,2)\} \leftarrow S(1,1), A(2,1)$$
$$\{A(2,1)\} \leftarrow [\![\mathsf{v}_2 \mapsto a]\!] \quad \{S(2,1)\} \leftarrow [\![\mathsf{v}_2 \mapsto b]\!] \quad \{\omega_{AS}(1,2)\} \leftarrow A(1,1), S(2,1)$$
$$\{S(1,2)\} \leftarrow \omega_{SA}(1,2) \quad \{S(1,2)\} \leftarrow \omega_{AS}(1,2) \quad \bot \leftarrow not\ S(1,2)$$
$$\bot \leftarrow \omega_{SA}(1,2), not\ S(1,2) \quad \bot \leftarrow A(1,1), not\ \omega_{AS}(1,2) \quad \bot \leftarrow S(1,1), not\ \omega_{SA}(1,2)$$
$$\bot \leftarrow \omega_{AS}(1,2), not\ S(1,2) \quad \bot \leftarrow A(2,1), not\ \omega_{SA}(1,2) \quad \bot \leftarrow S(2,1), not\ \omega_{AS}(1,2)$$
$$\bot \leftarrow [\![\mathsf{v}_1 \mapsto a]\!], not\ A(1,1) \quad \bot \leftarrow [\![\mathsf{v}_1 \mapsto b]\!], not\ S(1,1) \quad \bot \leftarrow [\![\mathsf{v}_1 \mapsto c]\!]$$
$$\bot \leftarrow [\![\mathsf{v}_2 \mapsto a]\!], not\ A(2,1) \quad \bot \leftarrow [\![\mathsf{v}_2 \mapsto b]\!], not\ S(2,1) \quad \bot \leftarrow [\![\mathsf{v}_2 \mapsto c]\!]$$

Unit-propagation on $\Delta(encode^{DC}(\mathcal{G}) \cup encode([\mathsf{v}_1, \mathsf{v}_2]))$ prunes the value c from the domains, i.e., sets $[\![\mathsf{v}_1 \mapsto c]\!]$ and $[\![\mathsf{v}_2 \mapsto c]\!]$ to false.

Unit-propagation of an ASP solver provides an efficient propagator for free, i.e., unit-propagation on our revised encoding is enough to achieve domain consistency.

▶ **Theorem 4.** *The assignments satisfying* GRAMMAR$(\mathcal{G}, [\mathsf{v}_1, .., \mathsf{v}_n])$ *correspond one-to-one to the answer sets of* $encode^{DC}(\mathcal{G}) \cup encode([\mathsf{v}_1, .., \mathsf{v}_n])$. *Unit-propagation on* $\Delta(encode^{DC}(\mathcal{G}) \cup encode([\mathsf{v}_1, .., \mathsf{v}_n]))$ *enforces domain consistency on* GRAMMAR$(\mathcal{G}, [\mathsf{v}_1, .., \mathsf{v}_n])$ *in* $\mathcal{O}(|P|n^3)$ *down any branch of the search tree.*

The proof follows the one in [26], where an and-or-graph is created that is very similar to the structure (i.e., the body-atom graph [20]) of $encode^{DC}(\mathcal{G})$, and subsequently, a SAT formula is constructed in a fashion that resembles the Clark's completion of our ASP model. Note that the propagators for the GRAMMAR constraint presented in [27, 25] have a similar overall complexity.

An extension which is sometimes useful in practice but goes slightly outside CFGs considers restrictions on some of the productions [26], e.g., in *shift scheduling* we want that *an employee works on an activity for a minimum of one hour*. Then, for a production of the form $A ::= BC$ and conditions represented by $\mathrm{c}_A(i,j), \mathrm{c}_B(i,j), \mathrm{c}_C(i,j)$ we restrict its application by encoding $A ::= BC$ in $encode(\mathcal{G})$ by

$$A(i,j) \leftarrow B(i,k), C(i{+}k, j{-}k), \mathrm{c}_A(i,j), \mathrm{c}_B(i,k), \mathrm{c}_C(i{+}k, j{-}k)\ .$$

This rule encodes that the nonterminal A generates a string starting at the ith symbol of length j if (1) the condition $\mathrm{c}_A(i,j)$ is satisfied, (2) B generates a string starting at the ith symbol of length k such that the condition $\mathrm{c}_B(i,k)$ is satisfied, and (3) C generates a string starting at the $i{+}k$th symbol of length $j{-}k$ such that the condition $\mathrm{c}_C(i{+}k, j{-}k)$ is satisfied. Similarly, productions of the form $A ::= t$ can be constrained. The changes to $encode^{DC}(\mathcal{G})$ are symmetric.

4 Modelling the Regular Constraint

In some cases, we only need a regular language, e.g., generated by a regular grammar, to specify problem constraints. One important point about regular grammars is that each nonterminal $A \in T[i,j]$ is derived by a substring at the ith symbol to the nth symbol, i.e., $j = n$. Using this insight we encode a production of the form $A ::= tB$ in $encode(\mathcal{G})$ as

below, for $1 \leq i \leq n$.

$A(i, n-i+1) \leftarrow [\![\mathsf{v}_i \mapsto t]\!], B(i+1, n-i)$

The changes to $encode^{DC}(\mathcal{G})$ are symmetric. For regular languages, propagation is faster.

▶ **Theorem 5.** *If $\mathcal{G}$ is regular then unit-propagation on $\Delta(encode^{DC}(\mathcal{G}) \cup encode([\mathsf{v}_1, .., \mathsf{v}_n]))$ enforces domain consistency on* GRAMMAR$(\mathcal{G}, [\mathsf{v}_1, .., \mathsf{v}_n])$ *in $\mathcal{O}(|P|n)$ down any branch of the search tree.*

Recall that each regular language $\mathcal{L}$ can also be specified by means of a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts assignments to a sequence of variables iff it is a member of $\mathcal{L}$. One important advantage of using an automata-based representation is that it permits to compress the ASP model using standard techniques for automaton minimisation. An automata-based propagator for the REGULAR constraint was first proposed in [23], and a SAT model such that unit-propagation enforces domain consistency was presented in [2]. We show here how to model REGULAR$(M, [\mathsf{v}_1, .., \mathsf{v}_n])$ with ASP in a straightforward and easily maintainable way. Given M, we propose an ASP model, denoted $encode(M)$, which represents all possible states the DFA can be in after processing i symbols. An accepted input string must generate a sequence of transformations starting at q_0 and ending in some finite state. Our encoding is as follows:

1. We introduce atoms $q_k(i)$ for each step i of M's processing, $0 \leq i \leq n$, and each state $q_k \in Q$ to indicate whether M is in state q_k after having processed the first i symbols.
2. Each transition $\delta(q_j, t) = q_k$ is encoded as follows, for $1 \leq i \leq n$.

   $q_k(i) \leftarrow q_j(i-1), [\![\mathsf{v}_i \mapsto t]\!]$

   Intuitively, whenever M is in state q_j after having processed the first $i-1$ symbols and M reads t as the ith symbol then M transitions to the state q_k in step i.
3. The condition that M must start processing in starting state q_0 is captured by

   $q_0(0) \leftarrow$

   which sets $q_0(0)$ unconditionally to true.
4. To represent that M must not finish processing in a rejecting state $q_{rej} \in Q \setminus F$, we post

   $\bot \leftarrow q_{rej}(n)$.

   Intuitively, it expresses the condition that no answer set contains $q_{rej}(n)$ for $q_{rej} \in Q \setminus F$.

To ensure that unit-propagation prunes all possible values, support has to be encoded. We extend $encode(M)$ by one item, resulting in $encode^{DC}(M)$.

6. There is support for $\mathsf{v}_i \mapsto t_i$ whenever there exists a transition $\delta(q_j, t) = q_k$ from the state q_j to the state q_k at step i while reading t. To encode support for $\mathsf{v}_i \mapsto t$, we define auxiliary atoms $d(q_j, q_k, i)$ for each transition $\delta(q_j, t) = q_k$, for $1 \leq i \leq n$, by

   $d(q_j, q_k, i) \leftarrow q_j(i-1), q_k(i)$.

   Now, for each assignment $\mathsf{v}_i \mapsto t$, we encode the existence of such a support by

   $\bot \leftarrow [\![\mathsf{v}_i \mapsto t]\!], not\ d(q_{j_1}, q_{k_1}, i), not\ d(q_{j_2}, q_{k_2}, i), .., d(q_{j_m}, q_{k_m}, i)$.

   This rule is satisfied if either $\mathsf{v}_i \mapsto t$ has a support, or $[\![\mathsf{v}_i \mapsto t]\!]$ is false.

▶ **Theorem 6.** *The answer sets of $encode(M) \cup encode([\mathsf{v}_1, .., \mathsf{v}_n])$ correspond one-to-one to assignments that satisfy* REGULAR$(M, [\mathsf{v}_1, .., \mathsf{v}_n])$. *Unit-propagation on $\Delta(encode^{DC}(M) \cup encode([\mathsf{v}_1, .., \mathsf{v}_n]))$ enforces domain consistency on* REGULAR$(M, [\mathsf{v}_1, .., \mathsf{v}_n])$ *in $\mathcal{O}(|\delta|n)$ down any branch of the search tree.*

The proof follows the one in [2], i.e., we can exploit the fact that Clark's completion of our ASP model results in the SAT formula presented in [2]. Note that the propagator for the REGULAR constraint proposed in [23] has a similar overall complexity.

5 Modelling the Precedence Constraint

For breaking value symmetry in a CSP, we only need a special case of the REGULAR constraint. Recall, PRECEDENCE$([t_1,..,t_m],[\mathsf{v}_1,..,\mathsf{v}_n])$ holds iff whenever $\mathsf{v}_j \mapsto t_\ell$ then there exists $\mathsf{v} \mapsto t_k$ such that $i < j$, for all $1 \leq k < \ell \leq m$. We can model the PRECEDENCE constraint with ASP. Our encoding, denoted $encode(pre[t_1,..,t_m])$, is as follows:

1. We introduce new atoms $taken(t_\ell, j)$ for each $t_\ell \in [t_1,..,t_m]$, and each position j, $0 \leq j \leq n$, to indicate whether $\mathsf{v}_i \mapsto t_\ell$ for some $i < j$.
2. For each value $t_\ell \in [t_1,..,t_m]$, we encode that it has been taken if $\mathsf{v}_i \mapsto t_\ell$, $1 \leq i \leq n$.
   $taken(t_\ell, i{+}1) \leftarrow [\![\mathsf{v}_i \mapsto t_j]\!]$
   To propagate the information to the other indices greater than i, our ASP model contains
   $taken(t_\ell, i{+}1) \leftarrow taken(t_\ell, i)$.
3. Finally, we post the condition that v_j cannot be assigned a value t_ℓ such that $t_\ell \in [t_1,..,t_m]$ if some value t_k has not been taken by a variable v_i such that $i < j$, $1 \leq k < \ell \leq m$.
   $\bot \leftarrow [\![\mathsf{v}_j \mapsto t_\ell]\!], not\ taken(t_k, i)$

Compared to our encodings for REGULAR constraints, our ASP model for the PRECEDENCE constraint is more economical with respect to auxiliary variables and rule size while unit-propagation can still enforce domain consistency.

▶ **Theorem 7.** *The assignments satisfying* PRECEDENCE$([t_1,..,t_m],[\mathsf{v}_1,..,\mathsf{v}_n])$ *correspond one-to-one to the answer sets of* $encode(pre[t_1,..,t_m]) \cup encode([\mathsf{v}_1,..,\mathsf{v}_n])$. *Down any branch of the search tree, unit-propagation on* $\Delta(encode(pre[t_1,..,t_m]) \cup encode([\mathsf{v}_1,..,\mathsf{v}_n]))$ *enforces domain consistency on* PRECEDENCE$([t_1,..,t_m],[\mathsf{v}_1,..,\mathsf{v}_n])$ *in* $\mathcal{O}(m^2 n)$.

6 Experiments on Shift Scheduling

We tested the practical utility of our ASP models of the GRAMMAR constraint on a set of shift-scheduling benchmarks [7]. 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 represents a time slot of 15 minutes, that must be generated by the following CFG $\mathcal{G}$.

$$S ::= RFR, \mathrm{c}_F(i,j) \equiv 30 \leq j \leq 38 \quad F ::= PLP \quad L ::= lL \mid l, \mathrm{c}_L(i,j) \equiv j = 4$$
$$S ::= RPR, \mathrm{c}_F(i,j) \equiv 13 \leq j \leq 24 \quad P ::= WbW \quad W ::= A_i, \mathrm{c}_W(i,j) \equiv j \geq 4$$
$$A_i ::= a_i A_i \mid a_i, \mathrm{c}_{A_i}(i,j) \equiv open(i) \quad R ::= rR \mid r$$

Related work in [26] converts the *shift scheduling* problem into a SAT model, denoted SAT. Experiments also consider our two ASP models of the GRAMMAR constraint: $encode(\mathcal{G})$ and $encode^{DC}(\mathcal{G})$. We denote these models as ASP and ASP-DC, respectively. A bottom-up ASP grounder such as GRINGO (3.0.3; [24]) can be employed to instantiate our models. Then, the grounder simulates a CYK parser, i.e., it constructs all possible parsings for all possible subsequences of the input string. However, as the CYK parser, it also instantiates productions that cannot participate in a successful parsing, i.e., productions which are uninteresting for us. We have implemented a grounder for the special purpose of this benchmark based on

Table 1 Results on shift scheduling; $|A|$: number of activities; #: problem number; m: number of employees; number of worked hours (boldfaced if best solution found amongst the different methods)

$\|A\|$	#	m	ASP	ASP-DC	SAT	$\|A\|$	#	m	ASP	ASP-DC	SAT
1	2	4	**26.00**	26.25	**26.00**	2	1	5	**25.00**	**25.00**	**25.00**
1	3	6	**37.25**	37.50	37.50	2	2	10	**58.00**	58.75	59.25
1	4	6	**38.00**	**38.00**	**38.00**	2	3	6	**39.50**	40.25	**39.50**
1	5	5	**24.00**	**24.00**	**24.00**	2	4	11	**68.25**	68.50	69.00
1	6	6	**33.00**	**33.00**	**33.00**	2	5	4	**24.50**	24.75	25.50
1	7	8	**49.00**	**49.00**	**49.00**	2	6	5	**28.25**	29.25	28.50
1	8	3	**20.50**	**20.50**	**20.50**	2	8	5	**32.00**	32.75	32.25
1	10	9	**54.00**	54.25	54.25	2	9	3	**19.00**	**19.00**	**19.00**
						2	10	8	57.25	57.75	**57.00**

the algorithm in [26]. This will allow for unit-propagation in the ASP model to achieve a strong type of local consistency close to domain consistency. Experiments were run with the system CLASP (1.3.5; [24]) on a 2.00 GHz PC under Linux, where each run was limited to 3600 sec time and 1 GB RAM. Table 1 presents the results for 17 satisfiable instances of the benchmark involving one or two activities. The solver returned a feasible solution for all instances regardless of the model after a few seconds, and subsequently optimised the solution, where no model performs significantly better that the other. However, we can draw a few conclusions. First, the ASP model $encode(\mathcal{G})$ is strong enough in our setting, i.e., enforcing domain consistency does not increase runtime. And, second, we do not pay any penalty for using our straightforward, easily maintainable ASP models vs the SAT model.

7 Experiments on Graph Colouring

A *colouring* of a graph (V, E) is a a mapping c from V to $\{1, .., k\}$ such that $c(v) \neq c(w)$ for every edge $(v, w) \in E$ with a given number k of colours. The *graph colouring* problem is to determine the existence of a colouring. Our experiments consider different options for breaking value symmetry between the colours: The options REGULAR and REGULAR-DC use our ASP model for a DFA-based encoding of the PRECEDENCE constraint. The option ALL uses our ASP model for the PRECEDENCE constraint to break all value symmetry. We denote PAIRWISE our ASP model of the method [18] which posts PRECEDENCE constraints between pairwise interchangeable values. The option NONE breaks no symmetry while GENERIC employs the preprocessor SBASS (1.0; [24]) for symmetry breaking in terms of detected symmetry generators [10]. We experimented on random *graph colouring* instances, but restricted ourselves to 3-, 4- and 5-colourings, when we noticed that the relative performance of symmetry breaking increased with each additional colour (exponentially with the number of colours). For each of the k-colouring experiments we generated 600 instances around the phase transition density with 400, 150, 75 vertices, respectively. All tests were run with the system CLASP (1.3.2) on a 2.00 GHz PC under Linux, where each run was limited to 600 s time and 1 GB RAM. The results on 3-, and 5- colourings shown in Figure 1 indicate that all symmetry breaking techniques considered in our study are effective, i.e., improve runtime. The data on any colouring clearly suggest that the GENERIC and REGULAR methods are inferior to enforcing value precedence through the PRECEDENCE constraint using PAIRWISE and ALL, or REGULAR-DC, where REGULAR-DC outperforms REGULAR due to the strong type of local consistency achieved in REGULAR-DC. For the 3-colouring case, ALL gives a significantly better improvement compared to PAIRWISE. For the 5-colouring case the

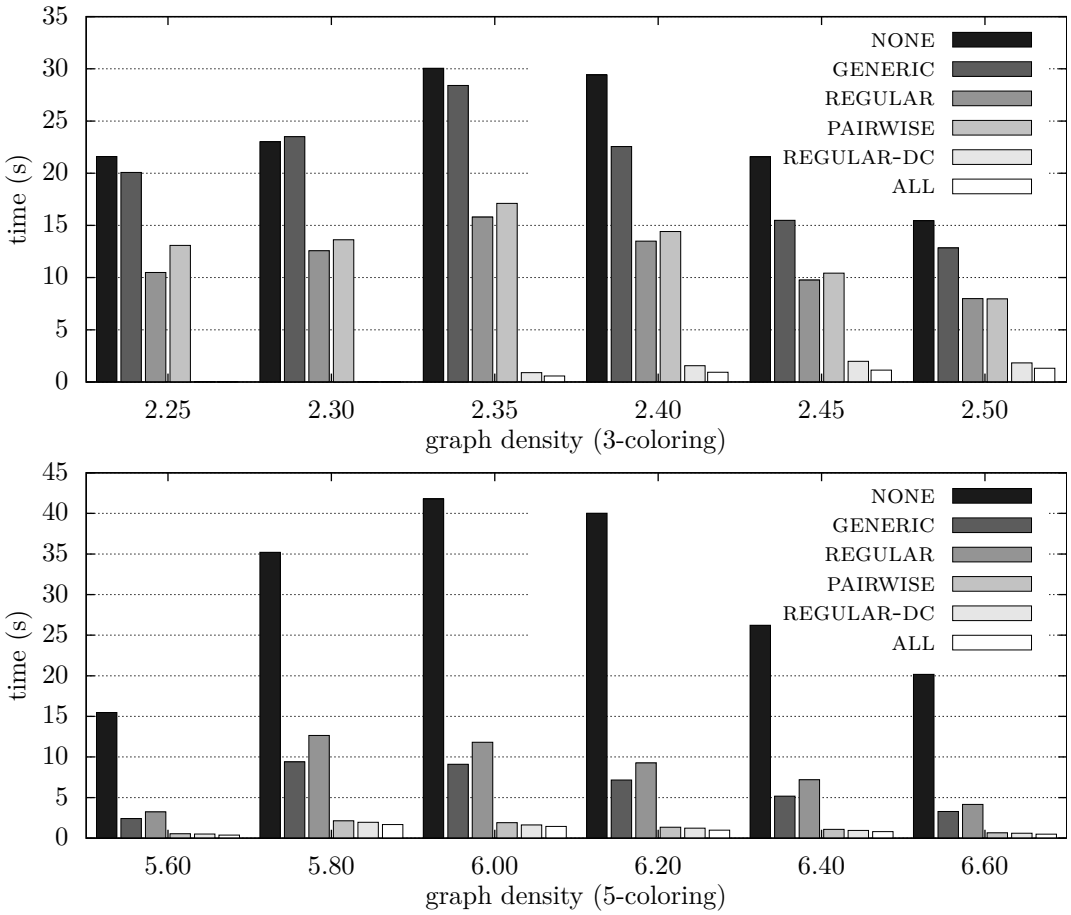

Figure 1 Histogram of the average time required by different symmetry breaking approaches.

same conclusion can be drawn, albeit less convincing. (5-colouring instances have fewer vertices, i.e., variables. This can improve propagation between PRECEDENCE constraints in PAIRWISE.) The overall conclusion from our graph colouring experiments is that breaking all value symmetry enforcing *precedence* is most effective.

8 Conclusions

Our work addresses modelling and solving CSP with ASP. We have presented ASP models of GRAMMAR and related constraints specified via grammars or automata. Finally, we have given an ASP model for the PRECEDENCE constraint. All our encodings are straightforward and easily maintainable without paying any penalty vs related SAT models. Unit-propagation of an ASP solver can prune all values, i.e., unit-propagation can achieve domain consistency on the original constraint. Future work concerns *lazy* modelling techniques [22] and ASP encodings of further constraints useful for modelling and solving CSP.

Acknowledgements We would like to thank Claude-Guy Quimper and Nina Narodytska for providing us with the benchmark data. NICTA is funded by the Department of Broadband, Communications and the Digital Economy, and the Australian Research Council.

References

1 R. Axelsson, K. Heljanko, and M. Lange. Analyzing context-free grammars using an incremental sat solver. In *Proceedings of ICALP'08*, pages 410–422, 2008.

2 F. Bacchus. GAC via unit propagation. In *Procedings of CP'07*, pages 133–147. Springer, 2007.

3 C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

4 A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009.

5 N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

6 K. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

7 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.

8 R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

9 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.

10 C. Drescher, O. Tifrea, and T. Walsh. Symmetry-breaking answer set solving. In *Proceedings of ICLP'10, ASPOCP'10 Workshop*, 2010.

11 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.

12 E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3(4-5):499–518, 2003.

13 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, 2007.

14 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

15 X. Jia, J.-H. You, and L.-Y. Yuan. Adding domain dependent knowledge into answer set programs for planning. In *Proceedings of ICLP'04*, pages 400–415. Springer, 2004.

16 G. Katsirelos, S. Maneth, N. Narodytska, and T. Walsh. Restricted global grammar constraints. In *Proceedings of CP'09*, pages 501–508. Springer, 2009.

17 G. Katsirelos, N. Narodytska, and T. Walsh. Reformulating global grammar constraints. In *Proceedings of CPAIOR'09*, pages 132–147. Springer, 2009.

18 Y. C. Law and J. H.-M. Lee. Global constraints for integer and set value precedence. In *Proceedings of CP'04*, pages 362–376. Springer, 2004.

19 J. Lee. A model-theoretic counterpart of loop formulas. In *Proceedings of IJCAI'05*, pages 503–508. Professional Book Center, 2005.

20 T. Linke. Suitable graphs for answer set programming. In *Proceedings of ASP'03*, pages 15–28, 2003.

21 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.

22 O. Ohrimenko and P. J. Stuckey. Modelling for lazy clause generation. In *CATS*, pages 27–37. Australian Computer Society, 2008.

23 G. Pesant. A regular language membership constraint for finite sequences of variables. In *Proceedings of CP'04*, pages 482–495. Springer, 2004.

24 `http://potassco.sourceforge.net/`.

25 C.-G. Quimper and T. Walsh. Global grammar constraints. In *Proceedings of CP'06*, pages 751–755. Springer, 2006.

26 C.-G. Quimper and T. Walsh. Decompositions of grammar constraints. In *Proceedings of AAAI'08*, pages 1567–1570. AAAI Press, 2008.

27 M. Sellmann. The theory of grammar constraints. In *Proceedings of CP'06*, pages 530–544. Springer, 2006.

28 P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

29 D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):372–375, 1967.