Conflict-Driven Constraint Answer Set Solving with Lazy Nogood Generation

Christian Drescher and **Toby Walsh**
NICTA and University of New South Wales, Locked Bag 6016, Sydney NSW 1466, Australia
Phone: +61-2-83060457, Email: {christian.drescher,toby.walsh}@nicta.com.au

Abstract

We present a new approach to enhancing answer set programming (ASP) with constraint programming (CP) techniques based on conflict-driven learning and lazy nogood generation.

Introduction

ASP has been put forward as a powerful paradigm to solve constraint satisfaction problems (CSP), i.e., empirical comparisons with CP has shown that, whilst ASP encodings are often highly competitive, non-propositional constructs like constraints over finite domains are more efficiently handled by CP systems (cf.Dovier, Formisano, and Pontelli, 2005). This led to the integration of CP techniques into ASP. Similar to satisfiability modulo theories, the key idea is to incorporate theory-specific predicates into propositional formulas, and extending an ASP solver's decision engine for a more high-level proof procedure. A promising approach to constraint answer set programming (CASP) has been presented in (Gebser, Ostrowski, and Schaub 2009) embedding a CP solver into an ASP solver and adding support for advanced backjumping and conflict-driven learning techniques. However, the elaboration of constraint interdependencies is limited by the poor interface between the ASP and the CP solver. To tackle this weakness, our work contributes constraint propagation via lazy nogood generation.

Constraint Answer Set Programming

Given a set of (constraint) variables V (each $v \in V$ has an associated finite domain $dom(v)$), a *(constraint logic) program* Π over an alphabet distinguishing regular atoms $\mathcal{A}$ and constraint atoms $\mathcal{C}$ is a finite set of rules of the form

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

where $a_0 \in \mathcal{A}$, $a_i \in \mathcal{A} \cup \mathcal{C}$ for $1 \leq i \leq n$, and $not\ a$ is the default negation of an atom a. For a rule r, define $body(r) = \{a_1, \dots, a_m, not\ a_{m+1}, \dots, not\ a_n\}$. The set of atoms occurring in Π is denoted by $atom(\Pi)$, the set of constraint atoms occurring in Π is denoted by $con(\Pi)$, and the set of bodies in Π is $body(\Pi) = \{body(r) \mid r \in \Pi\}$. Constraint atoms are identified with constraints via a function φ. The semantics of a program is given by its constraint answer sets. We refer the reader to (Gebser, Ostrowski, and Schaub 2009) for a comprehensive introduction to the semantics of CASP. In our approach to constraint answer set solving, an ASP solver deals with the atomic structure of the program, while a CP solver manages the constraints associated with constraint atoms. The variables, however, are tightly integrated with the ASP solver: Each possible variable assignment of the form $v = d$ or $v \leq d$ for $v \in V$ and $d \in dom(V)$ is represented as a regular atom from a distinguished alphabet $\mathcal{V}$, while possible variable assignments are identified with their representatives via a function ψ for providing a CP solver access to the domains of V. Rules have to be added to Π in order to enforce a node consistent set of variable assignments (cf. Drescher and Walsh, 2010). Then, constraint answer sets of the resulting program can be characterized via Boolean assignments over $atom(\Pi) \cup body(\Pi)$ that do not violate a set of nogoods imposed by Π. Formally, a Boolean *assignment* $\mathbf{A}$ is a sequence $(\sigma_1, \dots, \sigma_n)$ of *(signed) literals* σ_i of the form $\mathbf{T}a$ or $\mathbf{F}a$ where $a \in dom(\mathbf{A})$. The complement of a literal σ is denoted $\overline{\sigma}$. We access true and false objects in $\mathbf{A}$ via $\mathbf{A^T}$ and $\mathbf{A^F}$. A *nogood* represents a set $\delta = \{\sigma_1, \dots, \sigma_n\}$ of signed literals, expressing a constraint violated by any assignment $\mathbf{A}$ such that $\delta \subseteq \mathbf{A}$. We denote by Δ_Π the nogoods derived from the completion of Π, and by Λ_Π the ones given through loop formulas (Gebser et al. 2007). Similarly, we denote by Λ_c the nogoods that characterize $\phi(c)$ for $c \in con(\Pi)$. The pair $(\mathbf{A} \cap atom(\Pi), \psi^{-1}(\mathbf{A^T} \cap \mathcal{V}))$ is a constraint answer set of Π iff $\mathbf{A^T} \cup \mathbf{A^F} = dom(\mathbf{A})$, $\mathbf{A^F} \cap \mathbf{A^T} = \emptyset$, $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta_\Pi \cup \Lambda_\Pi \cup \bigcup_{c \in con(\Pi)} \Lambda_c$.

CDNL with Lazy Nogood Generation

Our decision procedure for CASP is based on the one for ASP called conflict-driven nogood learning (CDNL; Gebser et al., 2007). It includes conflict-driven learning and backjumping according to the 1UIP scheme (Mitchell 2005), but in contrast to CDNL we integrate external constraint propagators in form of lazy nogood generators to deal with non-propositional constraints. Given an updated variable assignment, a lazy nogood generator passes nogoods describing the inferences of the underlying constraint propagator to the ASP solver, similar to the principle of lazy clause generation (Ohrimenko, Stuckey and Codish 2009). Our main procedure is shown in Algorithm 1. It adds to the given pro-

Algorithm 1: CDNL-CASP

```
Input  : A constraint logic program $\Pi$, variables $V$.
Output: A constraint answer set of $\Pi$.
1  $\mathbf{A} \leftarrow \emptyset$                         // Boolean assigment
2  $\nabla \leftarrow \emptyset$                         // set of dynamic nogoods
3  $dl \leftarrow 0$                                   // decision level
4  $\Pi_V \leftarrow$ ENCODE($V, \Pi$)
5  loop
6      $(\mathbf{A}, \nabla) \leftarrow$ PROPAGATION($\Pi_V, \nabla, \mathbf{A}$)
7      if $\delta \subseteq \mathbf{A}$ for some $\delta \in \Delta_{\Pi_V} \cup \nabla$ then
8          if $dl = 0$ then return no answer set
9          $(\varepsilon, k) \leftarrow$ ANALYSIS($\delta, \Pi_V, \nabla, \mathbf{A}$)
10         $\nabla \leftarrow \nabla \cup \{\varepsilon\}$
11         $\mathbf{A} \leftarrow \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dl(\sigma)\}$
12         $dl \leftarrow k$
13     else if $\mathbf{A^T} \cup \mathbf{A^F} = atom(\Pi_V) \cup body(\Pi_V)$ then
14         return $(\mathbf{A^T} \cap \mathcal{A}, \psi^{-1}(\mathbf{A^T} \cap \mathcal{V}))$
15     else
16         $\sigma_d \leftarrow$ SELECT($\Pi_V, \nabla, \mathbf{A}$)
17         $dl \leftarrow dl + 1$
18         $\mathbf{A} \leftarrow \mathbf{A} \circ (\sigma_d)$
```

Algorithm 2: PROPAGATION

```
Input  : A program $P$, dyn. nogoods $\nabla$, assignment $\mathbf{A}$.
Output: An extended assignment and set of nogoods.
1  loop
2      repeat                                      // unit propagation
3          if $\delta \subseteq \mathbf{A}$ for some $\delta \in \Delta_\Pi \cup \nabla$ then
4              return $(\mathbf{A}, \nabla)$
5          $\Sigma \leftarrow \{\delta \in \Delta_\Pi \cup \nabla \mid \delta \setminus \mathbf{A} = \{\sigma\}, \overline{\sigma} \notin \mathbf{A}\}$
6          if $\Sigma \neq \emptyset$ then let $\sigma \in \delta \setminus \mathbf{A}$ for some $\delta \in \Sigma$ in
7              $\mathbf{A} \leftarrow \mathbf{A} \circ (\overline{\sigma})$
8      until $\Sigma = \emptyset$
9      $\varepsilon \leftarrow$ UNFOUNDEDSETPROPAGATION($\Pi, \mathbf{A}$)
10     if $\varepsilon = \emptyset$ then                        // no unfounded set
11         $\varepsilon \leftarrow$ CONSTRAINTPROPAGATION($\mathbf{A}|_\mathcal{V}, \mathbf{A}|_\mathcal{C}$)
12     if $\varepsilon = \emptyset$ then                    // nothing to propagate
13         return $(\mathbf{A}, \nabla)$
14     $\nabla \leftarrow \nabla \cup \{\varepsilon\}$
```

gram Π an encoding of consistent assignments to the variables in V (Line 4), and combines propagation (Line 6) with heuristic search (Lines 16–18). Conflict-driven learning is reflected by the dynamic nogoods in ∇, initialized in Line 2, and filled with learnt nogoods in Line 6 and 10. Lines 11–12 amount to backjumping guided by a decision level k determined by conflict analysis. Our algorithm returns a constraint answer set in form of an assignment to the regular atoms in Π and the variables in V, if a conflict-free total assignment was determined. We now explain our propagation procedure shown in Algorithm 2. (Observe that Π is instantiated by Π_V in Algorithm 1.) Lines 2–8 capture unit propagation (Mitchell 2005) on the nogoods in $\Delta_\Pi \cup \nabla$, resulting in either a conflict (Lines 3–4) or a fixpoint $\mathbf{A}$. The remaining Lines 9–14 amount to external propagators, i.e., the unfounded set check for addressing loop nogoods (Gebser et al. 2007), and constraint propagation (Lines 9,11). The unfounded set propagator returns a loop nogood $\varepsilon \in \Lambda_\Pi$ provided that a non-empty unfounded set has been determined, while the constraint propagator returns a nogood $\varepsilon \in \Lambda_c$ encoding its propagation provided a variable's domain was updated. The nogood ε is recorded in Line 14, triggering either unit propagation or conflict analysis.

Conclusions

We have outlined CDNL with lazy nogood generation as a hybrid approach to constraint answer set solving. Its key advantages are: First, conflict analysis can exploit constraint interdependencies which can improve propagation between constraints. Second, constraint propagation can contribute to search heuristics through lazy generated nogoods. Third, deletion strategies can be applied to lazy generated nogoods. We expect significant computational impact given the empirical evidence provided by lazy clause generation (Ohrimenko, Stuckey and Codish 2009). However, our approach abstracts the one from lazy clause generation: Every nogood can be syntactically represented by a clause, but other ASP constructs are also possible, such as cardinality and weight constraints. Future work includes the implementation of our hybrid system, and its experimental evaluation. A challenge that concerns the implementation is that all constraints are reified, but CP algorithms typically do not give any information about the negation of a constraint. However, they can be made to propagate weakly.

Acknowledgements NICTA is funded by the Department of Broadband, Communications and the Digital Economy, and the Australian Research Council.

References

Dovier, A.; Formisano, A.; and Pontelli, E. 2005. A comparison of CLP(FD) and ASP solutions to NP-complete problems. In *Proceedings of ICLP'05*, 67–82. Springer.

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

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *Proceedings of IJCAI'07*, 386–392. AAAI Press/MIT Press.

Gebser, M.; Ostrowski, M.; and Schaub, T. 2009. Constraint answer set solving. In *Proceedings of ICLP'09*, 235–249. Springer.

Mitchell, D. 2005. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science* 85:112–133.

Ohrimenko, O.; and Stuckey, P. J.; and Codish, M. 2009. Propagation via lazy clause generation. *Constraints* 14(3):357–391.