Stefano Bistarelli, Andrea Formisano, Marco Maratea (Eds.)

RCRA 2015

Proceedings of the 22nd RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion

Ferrara, Italy, September 22, 2015

Editors' address:
Università di Perugia
Dipartimento di Matematica e Informatica
Via Vanvitelli 1
06123 Perugia, Italy
{stefano.bistarelli | andrea.formisano}@unipg.it

Università di Genova
Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
viale F. Causa,15
16145 Genova, Italy
marco@dibris.unige.it

Program Chairs

Stefano Bistarelli	Università di Perugia
Andrea Formisano	Università di Perugia
Marco Maratea	Università di Genova

Program Committee

Mario Alviano	Università della Calabria
Paolo Baldan	Università di Padova
Marcello Balduccini	Drexel University
Laura Barbulescu	Carnegie Mellon University
Roman Bartak	Charles University in Prague
Elena Bellodi	Università di Ferrara
Stefano Bistarelli	Università di Perugia
Stefania Costantini	Università di L'Aquila
Agostino Dovier	Università di Udine
Andrea Formisano	Università di Perugia
Fabio Gadducci	Università di Pisa
Marco Gavanelli	Università di Ferrara
Matti Järvisalo	University of Helsinki
Zeynep Kiziltan	Università di Bologna
Ines Lynce	University of Lisbon
Daniele Magazzeni	King's College London
Toni Mancini	Sapienza Università Roma
Marco Maratea	Università di Genova
Joao Marques-Silva	University of Lisbon
Angelo Oddi	ISTI-CNR
Luca Pulina	Università di Sassari
Francesco Ricca	Università della Calabria
Francesco Santini	Università di Perugia
Torsten Schaub	University of Potsdam
Ivan Serina	Università di Brescia
Thomas Stützle	Université Libre de Bruxelles
Mauro Vallati	University of Huddersfield
Richard Wallace	University College Cork
Johannes Wallner	University of Helsinki

Contents

Regular Papers

Evaluating Answer Set Programming with Non-Convex Recursive Aggregates
Mario Alviano **1**

JWASP: A New Java-Based ASP Solver
Mario Alviano, Carmine Dodaro, and Francesco Ricca **16**

Searching for Sequential Plans Using Tabled Logic Programming
Roman Barták, Jindřich Vodrážka **24**

Testing Credulous and Sceptical Acceptance in Small-World Networks
Stefano Bistarelli, Fabio Rossi, Francesco Santini **39**

Now or Never: Negotiating Efficiently with Unknown Counterparts
Toni Mancini **47**

An Empirical Perspective on Ten Years of QBF Solving
Paolo Marin, Massimo Narizzano, Luca Pulina,
Armando Tacchella, Enrico Giunchiglia **62**

Modeling Abduction over Acyclic First-Order Logic Horn Theories in Answer Set Programming: Preliminary Experiments
Peter Schüller **76**

Papers not included here and published elsewhere

A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size
Mario Alviano, Carmine Dodaro, Francesco Ricca.
Appeared in Q. Yang and M. Wooldridge (Eds.): Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, pp. 2677–2683. AAAI Press

Fuzzy Answer Set Computation via Satisfiability Modulo Theories
Mario Alviano, Rafael Peñaloza.
Appeared in Theory and Practice of Logic Programming, 15(4-5), pp. 588-603, 2015. Cambridge University Press

Planning with Always Preferences by Compilation into STRIPS with Action Costs
Luca Ceriani, Alfonso E. Gerevini.
Appeared in L. Lelis and R. Stern (Eds.): Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, pp. 161-165. AAAI Press

ASCoL: a Tool for Improving Automatic Planning Domain Model Acquisition
Rabia Jilani, Andrew Crampton, Diane Kitchin, Mauro Vallati.
*Appeared in M. Gavanelli, E. Lamma, F. Riguzzi (Eds.): AI*IA 2015, Advances in Artificial Intelligence. Proceedings of the 14th International Conference of the Italian Association for Artificial Intelligence. LNCS 9336, 2015, pp. 438-451. Springer*

SyLVaaS: System Level Formal Verification as a Service
Toni Mancini, Federico Mari, Annalisa Massini, Igor Melatti, Enrico Tronci.
Appeared in M. Daneshtalab, M. Aldinucci, V. Leppänen, J. Lilius, M. Brorsson (Eds.): Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015, pp. 476-483. IEEE

Patient-Specific Models from Inter-Patient Biological Models and Clinical Records
Enrico Tronci, Toni Mancini, Ivano Salvo, Stefano Sinisi, Federico Mari, Igor Melatti, Annalisa Massini, Francesco Davi, Thomas Dierkes, Rainald Ehrig, Susanna Röblitz, Brigitte Leeners, Tillmann H. C. Kruger, Marcel Egli, Fabian Ille.
Appeared in K. Claessen and V. Kuncak (Eds.): Proceedings of Formal Methods in Computer-Aided Design, FMCAD 2014, pp. 207-214. IEEE

Identifying and Exploiting Features for Effective Plan Retrieval in Case-Based Planning
Mauro Vallati, Ivan Serina, Alessandro Saetti, Alfonso E. Gerevini.
Appeared in in R. Brafman, C. Domshlak, P. Haslum, and S. Zilberstein (Eds.): Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, pp. 239-243. AAAI Press

Evaluating Answer Set Programming with Non-Convex Recursive Aggregates

Mario Alviano

Department of Mathematics and Computer Science,
University of Calabria, 87036 Rende (CS), Italy
alviano@mat.unical.it

Abstract. Aggregation functions are widely used in answer set programming (ASP) for representing and reasoning on knowledge involving sets of objects collectively. These sets may also depend recursively on the results of the aggregation functions, even if so far the support for such recursive aggregations was quite limited in ASP systems. In fact, recursion over aggregates was restricted to convex aggregates, i.e., aggregates that may have only one transition from false to true, and one from true to false, in this specific order. Recently, such a restriction has been overcome, so that the user can finally use non-convex recursive aggregates in ASP programs, either on purpose or accidentally. A preliminary evaluation of ASP programs with non-convex recursive aggregates is reported in this paper.

Keywords: answer set programming, aggregation functions, non-convex recursive aggregates

1 Introduction

Answer set programming (ASP) is a declarative language for knowledge representation and reasoning [9]. ASP programs are sets of disjunctive logic rules, possibly using default negation under stable model semantics [21, 22]. Several constructs were added to the original, basic language in order to ease the representation of practical knowledge. Of particular interest are aggregate functions [5, 14, 17, 23, 27, 32], which allow for expressing properties on sets of atoms declaratively. In fact, in many ASP programs *functional dependencies* are enforced by means of COUNT aggregates, or equivalently using SUM aggregates; for example, a rule of the following form:

$$\bot \leftarrow R'(\overline{X}),\ \textsc{sum}[1, \overline{Y} : R(\overline{X}, \overline{Y}, \overline{Z})] \leq 1$$

constrains relation R to satisfy the functional dependency $\overline{X} \rightarrow \overline{Y}$, where $\overline{X} \cup \overline{Y} \cup \overline{Z}$ is the set of attributes of R, and R' is the projection of R on $\overline{X}$. Aggregate functions are also commonly used in ASP to constrain a nondeterministic guess. For example, in the *knapsack problem* the total weight of the selected items must not exceed a given limit, which can be modeled by the following rule:

$$\bot \leftarrow \textsc{sum}[W, O : \mathit{object}(O, W, C), \mathit{in}(O)] \leq \mathit{limit}.$$

Mainstream ASP solvers [15, 20] almost agree on the semantics of aggregates [14, 17], here referred to as F-stable model semantics, even if several valid alternatives were also considered in the literature [23, 30, 31, 33]. It is interesting to observe that F-stable model semantics was proposed more than a decade ago, providing a reasonable semantics for aggregates also in the recursive case. Indeed, it is based on an extension of the original program reduct, and on a minimality check of the stable model candidate resembling the disjunctive case. Despite this, for many years the implementation of F-stable model semantics was *incomplete*, and recursion over aggregates was restricted to *convex* aggregates [28], the largest class of aggregates for which the common reasoning tasks still belong to the first level of the polynomial hierarchy in the normal case [3]. In fact, convex aggregates may have only one transition from false to true, and one from true to false, in this specific order, a property that guarantees tractability of model checking in the normal case.

However, non-convex aggregations may arise in several contexts while modeling complex knowledge [1, 11, 13], and there are also minimalistic examples that are easily encoded in ASP using recursive non-convex aggregates, while alternative encodings not using aggregates are not so obvious. One of such examples is provided by the Σ_2^P-complete problem called *Generalized Subset Sum* [6]. In this problem, two vectors u and v of integers as well as an integer b are given, and the task is to decide whether the formula $\exists x \forall y(ux + vy \neq b)$ is true, where x and y are vectors of binary variables of the same length as u and v, respectively. For example, for $u = [1,2]$, $v = [2,3]$, and $b = 5$, the task is to decide whether the following formula is true: $\exists x_1 x_2 \forall y_1 y_2 (1 \cdot x_1 + 2 \cdot x_2 + 2 \cdot y_1 + 3 \cdot y_2 \neq 5)$. Any natural encoding of such an instance would include an aggregate of the form $\text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$. Luckily, a complete implementation of F-stable model semantics for common aggregation functions has been achieved this year by means of a translation combining disjunction and saturation in order to eliminate non-convexity from aggregates [4].

The aim of this paper is to evaluate a few problems that can be encoded in ASP using recursive non-convex aggregates. The tested programs are processed by the rewritings presented in [4], which are implemented in a prototype system written in Python that uses GRINGO and CLASP. In a nutshell, aggregates are represented by specific standard atoms, so that the grounding phase can be delegated to GRINGO [19], and the numeric output of GRINGO is then processed to properly encode aggregates for the subsequent stable model search performed by CLASP [20]. The focus of the paper is on programs using SUM aggregates, even if the tested system also supports several other common aggregation functions such as COUNT, AVG, MIN, MAX, EVEN, and ODD.

2 Background

Let $\mathcal{V}$ be a set of propositional atoms including $\perp$. A propositional literal is an atom possibly preceded by one or more occurrences of the *negation as failure* symbol $\sim$. An aggregate literal, or simply aggregate, is of the following form:

$$\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \odot b \tag{1}$$

where $n \geq 0$, $b, w_1, \ldots, w_n$ are integers, $l_1, \ldots, l_n$ are propositional literals, and $\odot \in \{<, \leq, \geq, >, =, \neq\}$. (Note that $[w_1 : l_1, \ldots, w_n : l_n]$ is a multiset.) A literal is either a propositional literal, or an aggregate. A rule r is of the following form:

$$p_1 \vee \cdots \vee p_m \leftarrow l_1 \wedge \cdots \wedge l_n \tag{2}$$

where $m \geq 1$, $n \geq 0$, $p_1, \ldots, p_m$ are propositional atoms, and $l_1, \ldots, l_n$ are literals. The set $\{p_1, \ldots, p_m\} \setminus \{\bot\}$ is referred to as head, denoted by $H(r)$, and the set $\{l_1, \ldots, l_n\}$ is called body, denoted by $B(r)$. A program $\mathit{\Pi}$ is a finite set of rules. The set of propositional atoms (different from $\bot$) occurring in a program $\mathit{\Pi}$ is denoted by $At(\mathit{\Pi})$, and the set of aggregates occurring in $\mathit{\Pi}$ is denoted by $Ag(\mathit{\Pi})$.

Example 1. Consider the following program $\mathit{\Pi}_1$:

$$x_1 \leftarrow {\sim}{\sim}x_1 \qquad x_2 \leftarrow {\sim}{\sim}x_2 \qquad y_1 \leftarrow \mathit{unequal} \qquad y_2 \leftarrow \mathit{unequal} \qquad \bot \leftarrow {\sim}\mathit{unequal}$$
$$\mathit{unequal} \leftarrow \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$$

As will be clarified after defining the notion of a stable model, $\mathit{\Pi}_1$ encodes the instance of Generalized Subset Sum introduced in Section 1. ■

An *interpretation* I is a set of propositional atoms such that $\bot \notin I$. Relation $\models$ is inductively defined as follows:

- for $p \in \mathcal{V}$, $I \models p$ if $p \in I$;
- $I \models {\sim}l$ if $I \not\models l$;
- $I \models \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \odot b$ if $\sum_{i \in [1..n], I \models l_i} w_i \odot b$;
- for a rule r, $I \models B(r)$ if $I \models l$ for all $l \in B(r)$, and $I \models r$ if $H(r) \cap I \neq \emptyset$ when $I \models B(r)$;
- for a program $\mathit{\Pi}$, $I \models \mathit{\Pi}$ if $I \models r$ for all $r \in \mathit{\Pi}$.

For any expression π, if $I \models \pi$, we say that I is a *model* of π, I satisfies π, or π is true in I. In the following, $\top$ will be a shorthand for ${\sim}\bot$, i.e., $\top$ is a literal true in all interpretations.

The *reduct* of a program $\mathit{\Pi}$ with respect to an interpretation I is obtained by removing rules with false bodies and by fixing the interpretation of all negative literals. More formally, the following function $F(I, \cdot)$ is inductively defined:

- for $p \in \mathcal{V}$, $F(I, p) := p$;
- $F(I, {\sim}l) := \top$ if $I \not\models l$, and $F(I, {\sim}l) := \bot$ otherwise;
- $F(I, \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \odot b) := \textsc{sum}[w_1 : F(I, l_1), \ldots, w_n : F(I, l_n)] \odot b$;
- for a rule r of the form (2), $F(I, r) := p_1 \vee \cdots \vee p_m \leftarrow F(I, l_1) \wedge \cdots \wedge F(I, l_n)$;
- for a program $\mathit{\Pi}$, $F(I, \mathit{\Pi}) := \{F(I, r) \mid r \in \mathit{\Pi}, I \models B(r)\}$.

Program $F(I, \mathit{\Pi})$ is the reduct of $\mathit{\Pi}$ with respect to I. An interpretation I is a *stable model* of a program $\mathit{\Pi}$ if $I \models \mathit{\Pi}$ and there is no $J \subset I$ such that $J \models F(I, \mathit{\Pi})$. Let $SM(\mathit{\Pi})$ denote the set of stable models of $\mathit{\Pi}$.

Example 2. Continuing with Example 1, the models of $\mathit{\Pi}_1$, restricted to the atoms in $At(\mathit{\Pi}_1)$, are X, $X \cup \{x_1\}$, $X \cup \{x_2\}$, and $X \cup \{x_1, x_2\}$, where $X = \{\mathit{unequal}, y_1, y_2\}$. Of these, only $X \cup \{x_1\}$ is a stable model. Indeed, the reduct $F(X \cup \{x_1\}, \mathit{\Pi}_1)$ is

$$x_1 \leftarrow \top \qquad y_1 \leftarrow \mathit{unequal} \qquad y_2 \leftarrow \mathit{unequal}$$
$$\mathit{unequal} \leftarrow \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$$

and no strict subset of $X \cup \{x_1\}$ is a model of the above program. On the other hand, the reduct $F(X \cup \{x_2\}, \mathit{\Pi}_1)$ is

$$\begin{array}{lll} x_2 \leftarrow \top & y_1 \leftarrow \mathit{unequal} & y_2 \leftarrow \mathit{unequal} \\ \mathit{unequal} \leftarrow \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5 & & \end{array}$$

and $\{x_2, y_2\}$ is a model of the above program. Similarly, it can be checked that X and $X \cup \{x_1, x_2\}$ are not stable models of $\mathit{\Pi}_1$. ■

An aggregate A is *convex* (in program reducts) if $J \models F(I, A)$ and $L \models F(I, A)$ implies $K \models F(I, A)$, for all $J \subseteq K \subseteq L \subseteq I \subseteq \mathcal{V}$. If A is convex then $I \models A$ and $J \models F(I, A)$ implies $K \models F(I, A)$, for all $J \subseteq K \subseteq I$. Note that aggregate $\textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$ from Example 1 is non-convex.

3 Non-Convex Aggregates Elimination

ASP solvers can only process sums of the form (1) in which all numbers are non-negative integers, and the comparison operator $\odot$ is $\geq$. This is due to the numeric format encoding the propositional program produced by the grounder. However, thanks to the rewritings proposed by [4], all sums can be rewritten in the form accepted by current ASP solvers. Following [4], strong equivalences can be used to restrict sums in the input program to only two forms, which are essentially (1) with $\odot \in \{\geq, \neq\}$. These first rewritings are given by means of strong equivalences [16, 25, 34].

Definition 1. *Let $\pi := l_1 \wedge \cdots \wedge l_n$ be a conjunction of literals, for some $n \geq 1$. A pair (J, I) of interpretations such that $J \subseteq I$ is an* SE-model *of π if $I \models \pi$ and $J \models F(I, l_1) \wedge \cdots \wedge F(I, l_n)$. Two conjunctions π, π' are* strongly equivalent, *denoted by $\pi \equiv_{SE} \pi'$, if they have the same SE-models.*

Strong equivalence means that replacing π by π' preserves the stable models of any logic program.

Proposition 1 (Lifschitz et al. 2001; Turner 2003; Ferraris 2005). *Let π, π' be two conjunctions of literals such that $\pi \equiv_{SE} \pi'$. Let $\mathit{\Pi}$ be a program, and $\mathit{\Pi}'$ be the program obtained from $\mathit{\Pi}$ by replacing any occurrence of π by π'. It holds that $\mathit{\Pi} \equiv_{\mathcal{V}} \mathit{\Pi}'$.*

The following strong equivalences can be proven by showing equivalence with respect to models, and by noting that $\sim$ is neither introduced nor eliminated:

(E1) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] > b \equiv_{SE} \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \geq b + 1$;
(E2) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \leq b \equiv_{SE} \textsc{sum}[-w_1 : l_1, \ldots, -w_n : l_n] \geq -b$;
(E3) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] < b \equiv_{SE} \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \leq b - 1$;
(E4) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] = b \equiv_{SE} \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \leq b \wedge$
$\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \geq b$.

For example, (E1) and (E3) are easy to obtain because b is integer by assumption. Similarly, (E4) is immediate by the semantics introduced in Section 2. For (E2), instead, the following equivalences can be observed:

(i) $I \models \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \leq b$;
(ii) $\sum_{i \in [1..n],\ I \models l_i} w_i \leq b$;
(iii) $\sum_{i \in [1..n],\ I \models l_i} -w_i \geq -b$;
(iv) $I \models \textsc{sum}[-w_1 : l_1, \ldots, -w_n : l_n] \geq -b$;

where (iii) above is obtained by multiplying both sides of the inequality (ii) by -1, and the equivalence of (i) and (ii), and of (iii) and (iv), is immediate by the semantics of sums. It is important to observe that the application of (E1)–(E4), from the last to the first, to a program $\mathit{\Pi}$ gives an equivalent program $pre(\mathit{\Pi})$ whose aggregates are sums with comparison operators $\geq$ and $\neq$.

Theorem 1. *Let $\mathit{\Pi}$ be a program. It holds that $\mathit{\Pi} \equiv_{\mathcal{V}} pre(\mathit{\Pi})$.*

After this preprocessing, the structure of the input program is further simplified by eliminating non-convex aggregates. To ease the presentation, and without loss of generality, hereinafter aggregates are assumed to be of the following form:

$$\begin{aligned} \textsc{sum}[&-w_1 : p_1, \ldots, -w_j : p_j, \\ &-w_{j+1} : {\sim}l_{j+1}, \ldots, -w_k : {\sim}l_k, \\ &\ w_{k+1} : p_{k+1}, \ldots, w_m : p_m, \\ &\ w_{m+1} : {\sim}l_{m+1}, \ldots, w_n : {\sim}l_n] \odot b \end{aligned} \tag{3}$$

where $n \geq m \geq k \geq j \geq 0$, $w_1, \ldots, w_n$ are positive integers, each p_i is a propositional atom, each l_i is a propositional literal, $\odot \in \{\geq, \neq\}$, and b is an integer. Intuitively, aggregated elements of (3) are partitioned in four sets, namely positive literals with negative weights, negative literals with negative weights, positive literals with positive weights, and negative literals with positive weights.

Let $\mathit{\Pi}$ be a program whose aggregates are of the form (3). Program $rew(\mathit{\Pi})$ is obtained from $\mathit{\Pi}$ by replacing each occurrence of an aggregate of the form (3) by a fresh, hidden propositional atom aux [10, 24]. Moreover, if $\odot$ is $\geq$, then the following rule is added:

$$\begin{aligned} aux \leftarrow \textsc{sum}[&w_1 : p_1^F, \ldots, w_j : p_j^F, \\ &w_{j+1} : {\sim}{\sim}l_{j+1}, \ldots, w_k : {\sim}{\sim}l_k, \\ &w_{k+1} : p_{k+1}, \ldots, w_m : p_m, \\ &w_{m+1} : {\sim}l_{m+1}, \ldots, w_n : {\sim}l_n] \geq b + w_1 + \cdots + w_k \end{aligned} \tag{4}$$

where each p_i^F is a fresh, hidden atom associated with the falsity of p_i, for all $i \in [1..j]$, and the following rules are also added to $rew(\mathit{\Pi})$:

$$p_i^F \leftarrow {\sim}p_i \tag{5}$$
$$p_i^F \leftarrow aux \tag{6}$$
$$p_i \vee p_i^F \leftarrow {\sim}{\sim}aux \tag{7}$$

Similarly, if $\odot$ is $\neq$, then the following rules are added to $rew(\mathit{\Pi})$:

$$\begin{aligned}
aux \leftarrow \text{SUM}[&w_1 : p_1^F, \ldots, w_j : p_j^F,\\
&w_{j+1} : {\sim}{\sim}l_{j+1}, \ldots, w_k : {\sim}{\sim}l_k,\\
&w_{k+1} : p_{k+1}, \ldots, w_m : p_m,\\
&w_{m+1} : {\sim}l_{m+1}, \ldots, w_n : {\sim}l_n] \geq b + 1 + w_1 + \cdots + w_k
\end{aligned} \tag{8}$$

$$\begin{aligned}
aux \leftarrow \text{SUM}[&w_1 : p_1, \ldots, w_j : p_j,\\
&w_{j+1} : {\sim}l_{j+1}, \ldots, w_k : {\sim}l_k,\\
&w_{k+1} : p_{k+1}^F, \ldots, w_m : p_m^F,\\
&w_{m+1} : {\sim}{\sim}l_{m+1}, \ldots, w_n : {\sim}{\sim}l_n] \geq -b + 1 + w_{k+1} + \cdots + w_n
\end{aligned} \tag{9}$$

together with rules (5)–(7) for each new p_i^F. Intuitively, any atom of the form p_i^F introduced by the rewriting must be true whenever p_i is false, but also when aux is true, so to implement what is usually referred to as *saturation* in the literature. Rules (5) and (6) encode such an intuition. Moreover, rule (7) guarantees that at least one between p_i and p_i^F belongs to any model of reducts obtained from interpretations containing aux. It is interesting to observe that when aux belongs to I the satisfaction of the associated aggregate can be tested according to all subsets of I in the reduct $F(\mathit{\Pi}, I)$.

The intuition behind (4) is that an interpretation I satisfies an aggregate of the form (3) such that $\odot$ is $\geq$ if and only if the following inequality is satisfied:

$$\sum_{i=1}^{j} -w_i \cdot I(p_i) + \sum_{i=j+1}^{k} -w_i \cdot I({\sim}l_i) + \sum_{i=k+1}^{m} w_i \cdot I(p_i) + \sum_{i=m+1}^{n} w_i \cdot I({\sim}l_i) \geq b \tag{10}$$

where $I(l) = 1$ if $I \models l$, and $I(l) = 0$ otherwise, for all literals l. Moreover, inequality (10) is satisfied if and only if the following inequality is satisfied:

$$\begin{aligned}
&\sum_{i=1}^{j} -w_i \cdot I(p_i) + \sum_{i=j+1}^{k} -w_i \cdot I({\sim}l_i) + \sum_{i=k+1}^{m} w_i \cdot I(p_i) +\\
&\qquad + \sum_{i=m+1}^{n} w_i \cdot I({\sim}l_i) + w_1 + \cdots + w_k \geq b + \sum_{i=1}^{k} w_i
\end{aligned} \tag{11}$$

and by distributivity (11) is equivalent to the following inequality:

$$\begin{aligned}
&\sum_{i=1}^{j} w_i \cdot (1 - I(p_i)) + \sum_{i=j+1}^{k} w_i \cdot (1 - I({\sim}l_i)) +\\
&\qquad + \sum_{i=k+1}^{m} w_i \cdot I(p_i) + \sum_{i=m+1}^{n} w_i \cdot I({\sim}l_i) \geq b + \sum_{i=1}^{k} w_i.
\end{aligned} \tag{12}$$

Note that $1 - I(l) = I({\sim}l)$ for all literals l, and p_i^F is associated with the falsity of p_i, for all $i \in [1..j]$. It is important to observe that negation was not used for positive literals in order to avoid oversimplifications in program reducts. Indeed, as already explained, for all $i \in [1..j]$, atom p_i^F will be derived true whenever p_i is false, but also when the aggregate is true.

The intuition behind (8)–(9) is similar. Essentially, an aggregate $\textsc{sum}(S) \neq b$ of the form (3) is true if and only if either $\textsc{sum}(S) \geq b + 1$ or $\textsc{sum}(S) \leq b - 1$ is true, and (E2) is applied to the second aggregate in order to use the previously explained rewriting. Let rew^* denote the composition $\mathit{rew} \circ \mathit{pre}$.

Example 3. Consider again program $\mathit{\Pi}_1$ from Example 1. Its rewriting $\mathit{rew}^*(\mathit{\Pi}_1)$ is as follows:

$$\begin{array}{lll}
x_1 \leftarrow {\sim}{\sim}x_1 \quad x_2 \leftarrow {\sim}{\sim}x_2 & y_1 \leftarrow \mathit{unequal} \quad y_2 \leftarrow \mathit{unequal} & \bot \leftarrow {\sim}\mathit{unequal} \\
\mathit{unequal} \leftarrow \mathit{aux} & \multicolumn{2}{l}{\mathit{aux} \leftarrow \textsc{sum}[1 : x_1^F; 2 : x_2^F; 2 : y_1^F; 3 : y_2^F] \geq 4} \\
 & \multicolumn{2}{l}{\mathit{aux} \leftarrow \textsc{sum}[1 : x_1; 2 : x_2; 2 : y_1; 3 : y_2] \geq 6} \\
x_1^F \leftarrow {\sim}x_1 & x_1^F \leftarrow \mathit{aux} & x_1 \vee x_1^F \leftarrow {\sim}{\sim}\mathit{aux} \\
x_2^F \leftarrow {\sim}x_2 & x_2^F \leftarrow \mathit{aux} & x_2 \vee x_2^F \leftarrow {\sim}{\sim}\mathit{aux} \\
y_1^F \leftarrow {\sim}y_1 & y_1^F \leftarrow \mathit{aux} & y_1 \vee y_1^F \leftarrow {\sim}{\sim}\mathit{aux} \\
y_2^F \leftarrow {\sim}y_2 & y_2^F \leftarrow \mathit{aux} & y_2 \vee y_2^F \leftarrow {\sim}{\sim}\mathit{aux}
\end{array}$$

The only stable model of $\mathit{rew}^*(\mathit{\Pi}_1)$ is $\{x_1, \mathit{unequal}, y_1, y_2, \mathit{aux}, x_1^F, x_2^F, y_1^F, y_2^F\}$. ■

Correctness of the rewriting can be established by slightly adapting the proof by [4].

Theorem 2 (Correctness). *Let $\mathit{\Pi}$ be a program. It holds that $\mathit{\Pi} \equiv_{At(\mathit{\Pi})} \mathit{rew}^*(\mathit{\Pi})$.*

4 Implementation

The rewritings introduced in Section 3 have been implemented in a prototype system written in Python and available at the following URL: `http://alviano.net/software/f-stable-models/`. The prototype accepts an input language whose syntax is almost conformant to ASP Core 2.0 [2]. It is a first-order language, meaning that propositional atoms are replaced by first-order atoms made of a predicate and a list or terms, where each term is an object constant, an object variable, or a composed term obtained by combining a function symbol with other terms. As usual in ASP, all variables are universally quantified, so that the propositional semantics given in Section 2 can be used after a grounding phase that replaces variables by constants in all possible ways.

The only exception to the ASP Core 2.0 format is that sums have to be encoded using the standard predicates f_sum and f_set. Moreover, only positive literals can occur in aggregation sets. In more detail, a sum of the form $\textsc{sum}[w_1 : p_1, \ldots, w_n : p_n] \odot b$, where $n \geq 0, b, w_1, \ldots, w_n$ are integers, $p_1, \ldots, p_n$ are (first-order) atoms, and $\odot \in \{<, \leq, \geq, >, =, \neq\}$ is encoded by the following first-order atom:

$$f\_sum(id, \mu(\odot), b)$$

where $\mu(\odot)$ equals `"<"`, `"<="`, `">="`, `">"`, `"="`, or `"!="`, and id is an identified for the aggregation set, encoded by the following rules:

$$f\_set(id, w_1, p_1) \leftarrow p_1 \qquad \cdots \qquad f\_set(id, w_n, p_n) \leftarrow p_n$$

where a body p_i ($i \in [1..n]$) can be omitted if p_i has no variables. (It is also possible to extend a body of the above rules in order to further constrain the aggregation set; for example, arithmetic expressions can be used to restrict the selection of atoms in the aggregation sets.)

Example 4. Program Π_1 from Example 1 is encoded as follows:

```
x_1 :- not not x_1.          unequal :- f_sum(uneq, "!=", 5).
x_2 :- not not x_2.          f_set(uneq, 1, x_1).
y_1 :- unequal.              f_set(uneq, 2, x_2).
y_2 :- unequal.              f_set(uneq, 2, y_1).
:- not unequal.              f_set(uneq, 3, y_2).
```

where `not` encodes the negation as failure symbol ∼, and rules with empty head are integrity constraints, i.e., rules whose head is equivalent to ⊥.

Alternatively, instances of Generalized Subset Sum can be specified by means of facts involving predicates $exists$, all, and $bound$. For example, the instance above is encoded by the following facts:

```
exists(x_1, 1).    all(y_1, 2).    bound(5).
exists(x_2, 2).    all(y_2, 3).
```

A program encoding the Generalized Subset Sum problem for instances encoded by these predicates is the following:

```
true(X, C) :- exists(X, C), not not true(X, C).
true(X, C) :- all(X, C), unequal.
:- not unequal.
unequal :- f_sum(uneq, "!=", B), bound(B).
f_set(uneq, C, true(X, C)) :- true(X, C).
```

where X, C, and B are object variables. ■

Given a program encoded as described above, the prototype obtains its propositional version by means of the grounder GRINGO. During the grounding phase, instances of predicate f_sum are considered *external*, i.e., they are assigned the truth value *undefined* in order to prevent their elimination. These instances and those of predicate f_set are identified and mapped in data structures of the prototype, so to have an internal representation of all sums occurring in the propositional program. The rewritings presented in Section 3 are then applied to these sums in order to eliminate any non-convexity. The new sums, and any additional rule introduced by the rewriting process, are added to the propositional program. Finally, the propositional program is printed to the standard output using the numeric format of GRINGO, so that CLASP can be used for computing its F-stable models, which eventually coincide with the F-stable models of the original program because additional atoms are hidden.

5 Experiment

The implemented rewritings were tested on a few domains that can be encoded using recursive sums. One of them is the Generalized Subset Sum problem presented in the introduction, which is of particular relevance in this experiment because its natural encoding in ASP requires a recursive non-convex sum. In fact, an ASP encoding for this problem that does not rely on recursive sums is not available, and therefore in this case the performance of the prototype was compared with an SMT encoding fed into Z3. The other two problems considered in this experiment are k-Clique-Coloring and 2-QBF, Σ_2^p-complete problems whose natural encodings in ASP do not rely on recursive sums. In these two cases, an alternative encoding using recursive sums can be obtained, even if usually paying an overhead on the running time. The aim of the experiment for these two problems is to evaluate such an overhead. All tested instances are available at the following URL: `http://archives.alviano.net/publications/2015/RCRA2015-experiment.zip`.

The experiment was run on an Intel Xeon CPU 2.4 GHz with 16 GB of RAM. CPU and memory usage were limited to 900 seconds and 15 GB, respectively. GRINGO, CLASP, and Z3 were tested with their default settings. Their performances were measured by PYRUNLIM (`http://alviano.net/software/pyrunlim/`). The results are reported in Table 1, where each row reports the number of instances and, for each tested ASP encoding, the number of solved instances, the average execution time and the average memory consumption. Data for Z3 are not reported in the table because it was run only on Generalized Subset Sum, discussed below.

Generalized Subset Sum [6]. Two vectors u and v of integers as well as an integer b are given, and the task is to decide whether the formula $\exists x \forall y(ux + vy \neq b)$ is true, where x and y are vectors of binary variables of the same length as u and v, respectively. For an instance such that $u = u_1, \ldots, u_m$ ($m \geq 1$) and $v = v_1, \ldots, v_n$ ($n \geq 1$) the following ASP encoding was tested (actually, its non-propositional version):

$$\begin{array}{ll}
x_i \leftarrow {\sim}{\sim}x_i & \forall i \in [1..m] \\
y_i \leftarrow \mathit{unequal} & \forall i \in [1..n] \\
\bot \leftarrow {\sim}\mathit{unequal} & \\
\mathit{unequal} \leftarrow \mathrm{SUM}[u_1 : x_1, \ldots, u_m : x_m, v_1 : y_1, \ldots, v_n : y_n] \neq b &
\end{array}$$

Table 1. Performance of GRINGO+CLASP (number of solved instances; average execution time in seconds; average memory consumption in MB).

			Aggregates			**Alternative**		
	Benchmark	**inst**	**sol**	**time**	**mem**	**sol**	**time**	**mem**
	Generalized Subset Sum	46	38	1.1	44	n/a	n/a	n/a
	k-Clique Coloring	60	60	177.2	863	60	20.9	205
2-QBF	Preprocessing Track	17	8	64.8	171	9	98.3	171
	QBFLib Track	32	1	0.1	101	1	0.1	102
	Application Track	48	13	126.1	45	19	22.7	45

As for Z3, the following SMT encoding was tested:

$$\begin{array}{l} \forall y_1 \cdots \forall y_n(\ \mathit{ite}(x_1, u_1, 0) + \cdots + \mathit{ite}(x_m, u_m, 0) + \\ \qquad\qquad \mathit{ite}(y_1, v_1, 0) + \cdots + \mathit{ite}(y_n, v_n, 0) \neq b) \end{array}$$

where $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$ are Boolean constants and variables, respectively, and $\mathit{ite}(\phi, t_1, t_2)$ is an *if-then-else* expression, i.e., its interpretation is t_1 if ϕ is true, and t_2 otherwise. As reported in the table, the ASP encoding leads to an excellent performance in many cases, with 38 solved instances and an average execution time of around 1.1 seconds. The performance achieved within the SMT encoding is instead less attractive, with only 14 solved instances and an average execution time of around 34.7 seconds. The tested ASP solver is also more efficient in memory, using 44 MB on average, while 148 MB are used by Z3 to solve the SMT instances. The reason of such different performances is that SMT is a more expressive language, allowing arbitrary alternations of quantifies, while in ASP at most one alternation can be simulated by means of saturation techniques. It turns out that ASP solvers can implement more optimized algorithms for problems on the second level of the polynomial hierarchy.

k-Clique-Coloring [29]. Given a graph $G = (V, E)$ with n nodes, and an integer $k \geq 2$, is possible to assign k colors to vertices in V such that each maximal clique K of G contains two vertices of different colors? The tested encoding using non-convex sums is reported below (again, its non-propositional version was actually tested).

$$\begin{array}{ll}
x_c \leftarrow {\sim}{\sim} x_c & \forall x \in V,\ \forall c \in [1..k] \\
\bot \leftarrow \textsc{sum}[1 : x_1, \ldots, 1 : x_k] \neq 1. & \forall x \in V \\
\bot \leftarrow {\sim}\mathit{saturate} & \\
\mathit{in}_x \vee \mathit{out}_x \leftarrow & \forall x \in V \\
\mathit{in}_x \leftarrow \mathit{saturate} & \forall x \in V \\
\mathit{out}_x \leftarrow \mathit{saturate} & \forall x \in V \\
\mathit{saturate} \leftarrow \mathit{in}_x, \mathit{in}_y & \forall x, y \in V,\ x \neq y,\ (x, y) \notin E \\
\mathit{saturate} \leftarrow \mathit{in}_x, \mathit{in}_y, x_c, y_d & \forall x, y \in V,\ \forall c, d \in [1..k],\ c \neq d \\
\mathit{saturate} \leftarrow \mathit{out}_x, \textsc{sum}[n : \mathit{saturate}, & \\
\qquad\qquad -1 : \mathit{in}_{y_1}, \ldots, -1 : \mathit{in}_{y_{n-1}}, & \\
\qquad\qquad 1 : \mathit{in}_{z_1}, \ldots, 1 : \mathit{in}_{z_j}] \geq 0 & \forall x \in V,\ \text{where} \\
 & \{y_1, \ldots, y_{n-1}\} = V \setminus \{x\}, \\
 & \{z_1, \ldots, z_j\} = \{z \mid (x, z) \in E\}
\end{array}$$

Intuitively, a color is assigned to each vertex, and the saturation is activated whenever one of the following conditions is verified:

- the guessed K contains two non-adjacent nodes, i.e., K is not a clique;
- the guessed K contains two nodes with different colors;
- there is a vertex $x \in V \setminus K$ such that x is adjacent to all vertices in K, i.e., K is not a maximal clique.

The alternative encoding not using recursive sums is obtained by replacing the last rule above with the following rule:

$$\mathit{saturate} \leftarrow \mathit{out}_x, \mathit{out}_{y_1}, \ldots, \mathit{out}_{y_j} \qquad \forall x \in V$$

where $\{y_1,\ldots,y_j\} = \{y \in V \setminus \{x\} \mid (x,y) \notin E\}$. Intuitively, in this case the third condition leading to saturate is the following:

- there is a vertex $x \in V \setminus K$ such that all vertices in V that are not adjacent to x do not belong to K, i.e., K is not a maximal clique.

For this problem, both encodings lead to solve all tested instances, which are the graphs submitted to the 4th ASP Competition [2] for the Graph Coloring problem. However, the overhead due to the use of recursive non-convex aggregates slows the computation down by a factor of 8, and also the memory consumption is around 4 times higher.

2-QBF. Given a 2-DNF $\exists\overline{x}\forall\overline{y}\phi$, is the formula valid? The tested encoding not using sums is the following:

$$\begin{array}{ll}
x \leftarrow {\sim}{\sim}x & \forall x \in \overline{x} \\
\bot \leftarrow {\sim}\mathit{saturate} & \\
y^T \vee y^F \leftarrow & \forall y \in \overline{y} \\
y^T \leftarrow \mathit{saturate} & \forall y \in \overline{y} \\
y^F \leftarrow \mathit{saturate} & \forall y \in \overline{y} \\
\mathit{saturate} \leftarrow \mu(l_1),\ldots,\mu(l_n) & \forall l_1 \wedge \cdots \wedge l_n \in \phi,\ n \geq 1
\end{array}$$

where $\mu(x) = x$ and $\mu(\neg x) = {\sim}x$ for all $x \in \overline{x}$, and $\mu(y) = y^T$ and $\mu(\neg y) = y^F$ for all $y \in \overline{y}$. An equivalent encoding using non-convex sums can be obtained by replacing all rules with y^T or y^F in the head with the following rules:

$$\begin{array}{ll}
y^T \leftarrow \mathrm{SUM}[1 : \mathit{saturate}, -1 : y^F] \geq 0 & \forall y \in \overline{y} \\
y^F \leftarrow \mathrm{SUM}[1 : \mathit{saturate}, -1 : y^T] \geq 0 & \forall y \in \overline{y}
\end{array}$$

The tested instances are all the 2-QBF instances in the QBF Gallery 2014 (`http://qbf.satisfiability.org/gallery/results.html`). Also in this case there is an overhead due to the unnatural use of non-convex sums. It impacts significantly on the Application Track, where the difference in terms of solved instances is 6.

6 Related Work

F-stable model semantics [14, 17] is implemented by widely-used ASP solvers [15, 20]. The original definition in [14, 17] is slightly different than the one provided in Section 2. In fact, propositional formulas can be arbitrarily nested in [17], while a more constrained structure is assumed in this paper in order to achieve an efficient implementation. On the other hand, double negation is not permitted in [14], even if it can be simulated by means of auxiliary atoms: a rule $p \leftarrow {\sim}{\sim}p$ can be equivalently encoded by using a fresh atom p^F and the following subprogram: $\{p \leftarrow {\sim}p^F,\ p^F \leftarrow {\sim}p\}$. Similarly, negated literals cannot occur in the aggregates considered by [14] but again can must be encoded by means of auxiliary atoms. Another difference with [14] is on negated aggregates, which are not permitted by the language considered in this paper because [17] and [14] actually assign different semantics to programs with negated aggregates. As a final remark, the reduct of [14] does not remove negated literals from satisfied bodies, which however are necessarily true in all counter-models because double negation is not allowed.

Techniques to rewrite logic programs with aggregates into equivalent programs with simpler aggregates were investigated in the literature right from the beginning [32]. In particular, rewritings into LPARSE-like programs, which differ from those presented in this paper, were considered in [26]. As a general comment, since disjunction is not considered in [26], all aggregates causing a jump from the first to the second level of the polynomial hierarchy are excluded a priori. This is the case for aggregates of the form $\mathrm{SUM}(S) \neq b$, $\mathrm{AVG}(S) \neq b$, and $\mathrm{COUNT}(S) \neq b$, as first noted by [33], but also for comparators other than $\neq$ when negative weights are involved. In fact, in [26] negative weights are eliminated by a rewriting similar to the one in (4), but negated literals are introduced instead of auxiliary atoms, which may lead to unintuitive results [18]. A different rewriting was presented by [17], whose output are programs with nested expressions, a construct that is not supported by current ASP systems. Other relevant rewriting techniques were proposed in [8, 7], and proved to be quite efficient in practice. However, these rewritings produce aggregate-free programs preserving F-stable models only in the stratified case, or if recursion is limited to convex aggregates. On the other hand, it is interesting to observe that the rewritings of [8, 7] are applicable to the output of the rewritings presented in this paper in order to completely eliminate aggregates, thus preserving F-stable models in general.

Several other stable model semantics were proposed for interpreting logic programs with aggregates. Many of these semantics rely on stability checks that are not based on minimality [30, 31, 33], and therefore the rewritings presented by [4] and recalled in Section 3 cannot be used for these semantics. A more recent proposal is based on a stability check that essentially eliminates aggregates from program reducts [23], and therefore the rewritings by [4] cannot help also in this case. Finally, there are other ASP constructs that are semantically close to aggregates, such as DL [13] and HEX [12] atoms, for interacting with external knowledge bases possibly expressed in different languages; as these constructs cannot be compactly reduced to sums in general, the rewritings by [4] do not apply to these languages as well.

7 Conclusion

ASP takes advantage of several constructs to ease the representation of complex knowledge. Aggregation functions are among the most commonly used constructs in ASP specifications. The rewritings proposed by [4] provide a concrete simplification of the structure of aggregations in input programs, so to improve the efficiency of low-level reasoners. Such rewritings are implemented in a prototype system, presented in this paper, which reported a reasonable performance on benchmarks for which more tailored encodings using disjunction in rule heads exist. More relevant, when such an aggregate-free encoding is unknown or untuitive, for example in the Generalized Subset Sum problem, the rewritings implemented in the prototype are particularly useful. Indeed, in this specific benchmark ASP solving significantly outperforms an alternative encoding in the more expressive language of SMT.

It must be remarked that this is only a preliminary evaluation of recursive non-convex aggregates in ASP. For the future, we plan to collect more encodings for problems that can be easily represented by using recursive non-convex aggregates, so to obtain a more suitable test suite for evaluating the efficiency of ASP solvers in presence of aggregations of this kind. Moreover, we will investigate alternative mappings of common aggregation functions into sums, with the aim of simplifying some of the rewritings by [4]. In particular, concerning ODD and EVEN, the rewritings by [4] are quadratic in size, and hence an interesting question to answer is whether there exist alternative rewritings of these aggregations whose sizes remain linear.

References

1. Abseher, M., Bliem, B., Charwat, G., Dusberger, F., Woltran, S.: Computing secure sets in graphs using answer set programming. In: Inclezan, D., Maratea, M. (eds.) Seventh International Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2014) (2014)
2. Alviano, M., Calimeri, F., Charwat, G., Dao-Tran, M., Dodaro, C., Ianni, G., Krennwallner, T., Kronegger, M., Oetsch, J., Pfandler, A., Pührer, J., Redl, C., Ricca, F., Schneider, P., Schwengerer, M., Spendier, L.K., Wallner, J.P., Xiao, G.: The fourth answer set programming competition: Preliminary report. In: Cabalar, P., Son, T.C. (eds.) LPNMR. pp. 42–53. LNCS (2013)
3. Alviano, M., Faber, W.: The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In: Cabalar, P., Son, T.C. (eds.) Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8148, pp. 67–72. Springer (2013), `http://dx.doi.org/10.1007/978-3-642-40564-8_7`
4. Alviano, M., Faber, W., Gebser, M.: Rewriting recursive aggregates in answer set programming: back to monotonicity. Theory and Practice of Logic Programming (2015), to appear
5. Bartholomew, M., Lee, J., Meng, Y.: First-order semantics of aggregates in answer set programming via modified circumscription. In: Logical Formalizations of Commonsense Reasoning, Papers from the 2011 AAAI Spring Symposium, Technical Report SS-11-06, Stanford, California, USA, March 21-23, 2011. AAAI (2011), `http://www.aaai.org/ocs/index.php/SSS/SSS11/paper/view/2472`
6. Berman, P., Karpinski, M., Larmore, L.L., Plandowski, W., Rytter, W.: On the complexity of pattern matching for highly compressed two-dimensional texts. J. Comput. Syst. Sci. 65(2), 332–350 (2002), `http://dx.doi.org/10.1006/jcss.2002.1852`
7. Bomanson, J., Gebser, M., Janhunen, T.: Improving the normalization of weight rules in answer set programs. In: Fermé, E., Leite, J. (eds.) JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8761, pp. 166–180. Springer (2014), `http://dx.doi.org/10.1007/978-3-319-11558-0_12`
8. Bomanson, J., Janhunen, T.: Normalizing cardinality rules using merging and sorting constructions. Lecture Notes in Computer Science, vol. 8148, pp. 187–199. Springer (2013), `http://dx.doi.org/10.1007/978-3-642-40564-8_19`
9. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011), `http://doi.acm.org/10.1145/2043174.2043195`

10. Eiter, T., Tompits, H., Woltran, S.: On solution correspondences in answer set programming. In: Kaelbling, L., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05). pp. 97–102. Professional Book Center (2005)
11. Eiter, T., Fink, M., Krennwallner, T., Redl, C.: Conflict-driven ASP solving with external sources. Theory and Practice of Logic Programming 12(4-5), 659–679 (2012), `http://dx.doi.org/10.1017/S1471068412000233`
12. Eiter, T., Fink, M., Krennwallner, T., Redl, C., Schüller, P.: Efficient hex-program evaluation based on unfounded sets. J. Artif. Intell. Res. (JAIR) 49, 269–321 (2014), `http://dx.doi.org/10.1613/jair.4175`
13. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. 172(12-13), 1495–1539 (2008), `http://dx.doi.org/10.1016/j.artint.2008.04.002`
14. Faber, W., Pfeifer, G., Leone, N.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. 175(1), 278–298 (2011), `http://dx.doi.org/10.1016/j.artint.2010.04.002`
15. Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., Ielpa, G.: Design and implementation of aggregate functions in the DLV system. Theory and Practice of Logic Programming 8(5-6), 545–580 (2008), `http://dx.doi.org/10.1017/S1471068408003323`
16. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05). Lecture Notes in Artificial Intelligence, vol. 3662, pp. 119–131. Springer-Verlag (2005)
17. Ferraris, P.: Logic programs with propositional connectives and aggregates. ACM Trans. Comput. Log. 12(4), 25 (2011), `http://doi.acm.org/10.1145/1970398.1970401`
18. Ferraris, P., Lifschitz, V.: Weight constraints as nested expressions. Theory and Practice of Logic Programming 5(1-2), 45–74 (2005), `http://dx.doi.org/10.1017/S1471068403001923`
19. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in *gringo* series 3. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6645, pp. 345–351. Springer (2011), `http://dx.doi.org/10.1007/978-3-642-20895-9_39`
20. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. 187, 52–89 (2012), `http://dx.doi.org/10.1016/j.artint.2012.04.001`
21. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988 (2 Volumes). pp. 1070–1080. MIT Press (1988)
22. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991), `http://dx.doi.org/10.1007/BF03037169`
23. Gelfond, M., Zhang, Y.: Vicious circle principle and logic programs with aggregates. Theory and Practice of Logic Programming 14(4-5), 587–601 (2014), `http://dx.doi.org/10.1017/S1471068414000222`
24. Janhunen, T., Niemelä, I.: Applying visible strong equivalence in answer-set program transformations. In: Erdem, E., Lee, J., Lierler, Y., Pearce, D. (eds.) Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz. Lecture Notes in Computer Science, vol. 7265, pp. 363–379. Springer-Verlag (2012)
25. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic 2(4), 526–541 (2001)

26. Liu, G., You, J.: Relating weight constraint and aggregate programs: Semantics and representation. Theory and Practice of Logic Programming 13(1), 1–31 (2013), `http://dx.doi.org/10.1017/S147106841100038X`
27. Liu, L., Pontelli, E., Son, T.C., Truszczynski, M.: Logic programs with abstract constraint atoms: The role of computations. Artif. Intell. 174(3-4), 295–315 (2010), `http://dx.doi.org/10.1016/j.artint.2009.11.016`
28. Liu, L., Truszczynski, M.: Properties and applications of programs with monotone and convex constraints. J. Artif. Intell. Res. (JAIR) 27, 299–334 (2006), `http://dx.doi.org/10.1613/jair.2009`
29. Marx, D.: Complexity of clique coloring and related problems. Theor. Comput. Sci. 412(29), 3487–3500 (2011)
30. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. Theory and Practice of Logic Programming 7(3), 301–353 (2007), `http://dx.doi.org/10.1017/S1471068406002973`
31. Shen, Y., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T., Deng, J.: FLP answer set semantics without circular justifications for general logic programs. Artif. Intell. 213, 1–41 (2014), `http://dx.doi.org/10.1016/j.artint.2014.05.001`
32. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. 138(1-2), 181–234 (2002), `http://dx.doi.org/10.1016/S0004-3702(02)00187-X`
33. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. Theory and Practice of Logic Programming 7(3), 355–375 (2007), `http://dx.doi.org/10.1017/S1471068406002936`
34. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. Theory and Practice of Logic Programming 3(4-5), 609–622 (2003)

JWASP: A New Java-Based ASP Solver

Mario Alviano, Carmine Dodaro, and Francesco Ricca

Department of Mathematics and Computer Science,
University of Calabria, 87036 Rende (CS), Italy
{alviano,dodaro,ricca}@mat.unical.it

Abstract. Answer Set Programming (ASP) is a well-known declarative programming language for knowledge representation and non-monotonic reasoning. ASP solvers are usually written in C/C++ with the aim of extremely optimizing their performance. Indeed, C/C++ allow for several low level optimizations, which however come at the price of a less portable implementation. This is a problem for some real world use cases which do not actually require an extremely efficient computation, but would benefit from a platform-independent and easily-deployable implementation. Motivated by such use cases, we develop JWASP, a new ASP solver written in Java and extending the open source library SAT4J in order to process ASP programs with atomic heads. We also report on a preliminary experiment assessing the performance of JWASP, whose results are encouraging: JWASP is a good candidate as an alternative ASP solver for platform-independent applications, which cannot rely on current ASP solvers.

1 Introduction

Answer Set Programming (ASP) [5] is a declarative programming paradigm, which has been proposed in the area of non-monotonic reasoning and logic programming. The idea of ASP is to represent a given computational problem by a logic program whose answer sets correspond to solutions, and then use a solver to find them [5]. The availability of solvers has made possible the application of ASP for solving complex problems arising in several areas [1, 6], including AI, knowledge representation and reasoning, databases, bioinformatics. Recently ASP has been also used to solve a number of industry-level applications [7, 21].

Answer set programming is computationally hard, and modern ASP solvers are usually based on one of two alternative approaches. The first of these approaches consists in implementing a native algorithm by adapting SAT solving techniques [22]. In particular, CDCL backtracking with learning, restarts, and conflict-driven heuristics is extended with ASP-specific propagation techniques such as support inference via Clark's completion, and well-founded inference via source pointers [23]. The second approach resorts on rewriting techniques into SAT formulas, which are then evaluated by an off the shelf SAT solver [13].

ASP solvers, like SAT solvers, are developed having in mind the (often well-deserved) goal of maximizing performance. For this reason, ASP solvers are usually written in C/C++, a programming language that is suited for implementing several low level optimizations, but at the price of a reduced portability. This is a problem for some real world use cases which do not actually require the highest available performance in computation, but would benefit from a platform-independent and easily-deployable implementation. For example, the iTravel system [20] takes advantage of some ASP-based web services implemented as Java servlets interacting with DLV [16] via the DLV WRAPPER API [19]. Usually, Java servlets are easily exportable as WAR archives, which are then deployable to different servers by simply copying the archives. Such a simplicity was not possible with the ASP-based web services because different versions of DLV were required for servers running different operating systems. A similar issue also affects the distribution of ASPIDE [9], an IDE for ASP developed in Java which must include different versions of an ASP solver for different operating systems. An ASP solver implemented in Java would simplify the distribution of ASPIDE, not preventing the possibility to run other ASP solvers written in C/C++ if needed.

If on the one hand Java provides all the means for implementing a platform-independent ASP solver, on the other hand the following questions have to be answered: How much overhead is introduced? Is the performance of an ASP solver written in Java acceptable when compared with state of the art ASP solvers? Motivated by the needs arising in different use cases, and in order to answer these two questions, we developed JWASP (`https://github.com/dodaro/jwasp.git`), a new ASP solver written in Java. JWASP is based on the open source library SAT4J [15]. In particular, JWASP extends SAT4J in order to process ASP programs with atomic heads.

A preliminary experiment assessing the performance of JWASP has been conducted on benchmarks from the previous ASP competitions [1, 6]. In particular, JWASP was compared with the following state of the art ASP solvers: the native CLASP 3.1.1 [11] and WASP [3]; the rewriting-based LP2SAT endowed by GLUCOSE [4]; and LP2SAT endowed by SAT4J [15]. The results are encouraging. In fact, even if JWASP cannot match the performance of CLASP, which is actually expected, it can compete with a prominent rewriting-based ASP solver using GLUCOSE. Our experiment highlights that JWASP is a good candidate as an alternative ASP solver for platform-independent applications, where conventional solvers cannot be used or might not be comfortably integrated.

2 Preliminaries

Syntax and semantics of propositional logic and propositional ASP are briefly introduced in this section.

2.1 Propositional Logic

Syntax. Let $\mathcal{A}$ be a fixed, countable set of (Boolean) variables, or (propositional) atoms, including $\bot$. A *literal* ℓ is either an atom a, or its negation $\neg a$. A *clause* is a set of literals representing a disjunction, and a propositional formula φ is a set of clauses representing a conjunction, i.e., only formulas in *conjunctive normal form* (CNF) are considered here.

Semantics. An interpretation I is a set of literals over atoms in $\mathcal{A} \setminus \{\bot\}$. Intuitively, literals in I are true, literals whose complement is in I are false and the remaining literals are undefined. An interpretation I is total if there are no undefined literals, otherwise I is partial. An interpretation I is inconsistent if for an atom a both a and $\neg a$ are in I. Relation $\models$ is inductively defined as follows: for $a \in \mathcal{A}$, $I \models a$ if $a \in I$, and $I \models \neg a$ if $\neg a \in I$; for a clause c, $I \models c$ if $I \models \ell$ for some $\ell \in c$; for a formula φ, $I \models \varphi$ if $I \models c$ for all $c \in \varphi$. If $I \models \varphi$ then I is a *model* of φ, I *satisfies* φ, and φ is true w.r.t. I. If $I \not\models \varphi$ then I is not a model of φ, I *violates* φ, and φ is false w.r.t. I. Similar for literals, and clauses. A formula φ is *satisfiable* if there is an interpretation I such that $I \models \varphi$; otherwise, φ is *unsatisfiable*.

Example 1. Consider the following formula φ:

$$\{a, \neg b\} \qquad \{b, \neg a\} \qquad \{\neg a\} \qquad \{c\} \qquad \{c, \neg b\}$$

φ is satisfiable and the interpretation $I = \{\neg a, \neg b, c\}$ is a model. $\lhd$

2.2 Answer Set Programming

Syntax. Let $\sim$ denote *negation as failure*. A *$\sim$-literal* (or just literal when clear from the context) is either an atom (a positive literal), or an atom preceded by $\sim$ (a negative literal). A logic program $\mathit{\Pi}$ is a finite set of rules of the following form:

$$a \leftarrow b_1, \ldots, b_k, \sim b_{k+1}, \ldots, \sim b_m \qquad (1)$$

where $m \geq 0$, and $a, b_1, \ldots, b_m$ are atoms in $\mathcal{A}$. For a rule r of the form (1), set $\{a\}$ is called *head* of r, and denoted $H(r)$; conjunction $b_1, \ldots, b_m, \sim b_{k+1}, \ldots, \sim b_m$ is named *body* of r, and denoted $B(r)$; the sets $\{b_1, \ldots, b_k\}$ and $\{b_{k+1}, \ldots, b_m\}$ of positive and negative literals in $B(r)$ are denoted $B^+(r)$ and $B^-(r)$, respectively. A *constraint* is a rule r such that $H(r) = \{\bot\}$.

Semantics. An interpretation I is a set of $\sim$-literals over atoms in $\mathcal{A} \setminus \{\bot\}$. Relation $\models$ is extended as follows: for a negative literal $\sim a$, $I \models \sim a$ if $\sim a \in I$; for a conjunction $\ell_1, \ldots, \ell_n$ ($n \geq 0$) of literals, $I \models \ell_1, \ldots, \ell_n$ if $I \models \ell_i$ for all $i \in [1..n]$; for a rule r, $I \models r$ if $H(r) \cap I \neq \emptyset$ whenever $I \models B(r)$; for a program $\mathit{\Pi}$, $I \models \mathit{\Pi}$ if $I \models r$ for all $r \in \mathit{\Pi}$. The definition of a stable model is based on a notion of program reduct [12]: Let $\mathit{\Pi}$ be a normal logic program, and I an interpretation. The reduct of $\mathit{\Pi}$ w.r.t. I, denoted $\mathit{\Pi}^I$, is obtained from $\mathit{\Pi}$ by deleting each rule r such that $B^-(r) \cap I \neq \emptyset$, and removing negative literals in the remaining rules. An interpretation I is an answer set for $\mathit{\Pi}$ if $I \models \mathit{\Pi}$ and there is no total interpretation J such that $J \cap \mathcal{A} \subset I \cap \mathcal{A}$ and $J \models \mathit{\Pi}^I$. The set of all answer sets of a program $\mathit{\Pi}$ is denoted $SM(\mathit{\Pi})$. Program $\mathit{\Pi}$ is *coherent* if $SM(\mathit{\Pi}) \neq \emptyset$; otherwise, $\mathit{\Pi}$ is *incoherent*.

Example 2. Consider the following program $\mathit{\Pi}$:

$$\begin{array}{lll} a \leftarrow c & a \leftarrow b, \sim e & b \leftarrow a, \sim e \\ c \leftarrow \sim d & d \leftarrow \sim c & e \leftarrow \sim d \end{array}$$

$I = \{a, \sim b, c, \sim d, e\}$ is an answer set of $\mathit{\Pi}$. $\lhd$

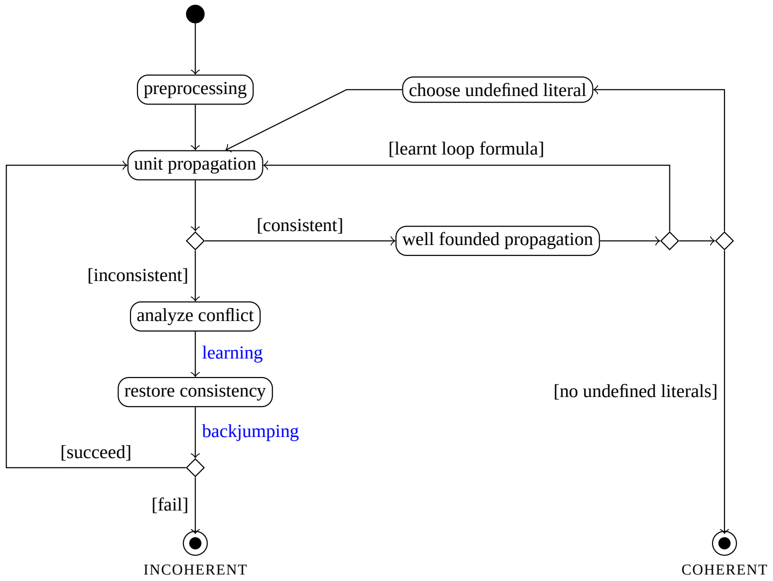


Fig. 1. Computation of an answer set in JWASP.

3 Answer Set Computation in JWASP

In this section we first review the algorithms implemented in JWASP for the computation of an answer set, and then we describe how these were implemented by extending SAT4J. The presentation is properly simplified to focus on the main principles.

3.1 Main Algorithms

The main algorithm is depicted in Figure 1.

Preprocessing. The first step is a preprocessing of the input program Π, that is transformed into a propositional formula called the *Clark's completion* of the program Π, denoted $Comp(\Pi)$. This step is performed since answer sets are supported models [17]. A model I of a program Π is *supported* if each $a \in I \cap \mathcal{A}$ is supported, i.e., there exists a rule $r \in \Pi$ such that $H(r) = a$, and $B(r) \subseteq I$. In more detail, given a rule $r \in \Pi$, let aux_r denote a fresh atom, i.e., an atom not appearing elsewhere, the completion of Π consists of the following clauses:

- $\{\neg a, aux_{r_1}, \ldots, aux_{r_n}\}$ for each atom a occurring in Π, where $r_1, \ldots, r_n$ are the rules of Π whose head is a;
- $\{H(r), \neg aux_r\}$ and $\{aux_r\} \cup \bigcup_{a \in B^+(r)} \neg a \cup \bigcup_{a \in B^-(r)} a$ for each rule $r \in \Pi$;
- $\{\neg aux_r, \ell\}$ for each $r \in \Pi$ and $\ell \in B(r)$.

After computing the Clark's completion $Comp(\Pi)$, the input is further simplified applying classical preprocessing techniques of SAT solvers [8], and then the nondeterministic search takes place.

CDCL Algorithm. The main ASP solving algorithm is similar to the CDCL procedure of SAT solvers. In the beginning a partial interpretation I is set to $\emptyset$. Function unit propagation extends I with those literals that can be deterministically inferred. This function returns false if an inconsistency (or conflict) is detected, true otherwise. When an inconsistency is detected, the algorithm analyzes the inconsistent interpretation and learns a clause using the *1-UIP* learning scheme [18]. The learned clause models the inconsistency in order to avoid exploring the same search branch several times. Then, the algorithm unrolls choices until consistency of I is restored, and the computation resumes by propagating the consequences of the clause learned by the conflict analysis. If the consistency cannot be restored, the algorithm terminates returning INCOHERENT. When no inconsistency is detected, the well founded propagation (detailed in the following) checks whether I is *unfounded-free*. In case I is not unfounded-free a clause is added to $Comp(\Pi)$ and unit propagation is invoked. If I is unfounded-free and the interpretation I is total then the algorithm terminates returning COHERENT and I is an answer set of Π. Otherwise, an undefined literal, say ℓ, is chosen according to some heuristic criterion. The computation then proceeds on $I \cup \{\ell\}$. Unit propagation and well founded propagation are described in more detail in the following.

Propagation rules. JWASP implements two deterministic inference rules for pruning the search space during answer set computation. These propagation rules are named *unit* and *well founded*. Unit propagation is applied first. It returns false if an inconsistency arises. Otherwise, well founded propagation is applied. Well founded propagation may learn an implicit clause in Π, in which case unit propagation is applied on the new clause. More in details, unit propagation is as in SAT solvers: An undefined literal ℓ is inferred by unit propagation if there is a clause c that can be satisfied only by ℓ, i.e., c is such that $\ell \in c$ is undefined and all literals in $c \setminus \{\ell\}$ are false w.r.t. I. Concerning well founded propagation, we must first introduce the notion of an unfounded set. A set X of atoms is *unfounded* if for each rule r such that $H(r) \cap X \neq \emptyset$, at least one of the following conditions is satisfied: (i) a literal $\ell \in B(r)$ is false w.r.t. I; (ii) $B^+(r) \cap X \neq \emptyset$. Intuitively, atoms in X can have support only by themselves. Well founded propagation checks whether the interpretation contains an unfounded set X. In this case, it learns a clause forcing falsity of an atom in X. Clauses for other atoms in X will be learned on subsequent calls to the function, unless an inconsistency arises during unit propagation. In case of inconsistencies, indeed, the unfounded set X is recomputed.

3.2 Implementation

The implementation of a modern and efficient ASP solver requires the implementation of at least three modules. The first module is the parser of a ground ASP program. The second module computes the Clark's completion. The third module implements the CDCL backtracking algorithm extended by applying well founded propagation as presented in Section 3.1. Concerning the parser, JWASP accepts as input normal ground programs expressed in the numeric format of GRINGO [10]. The Clark's completion is computed after the whole program has been parsed. The third module is implemented by JWASP exploiting the open source Java library SAT4J [15]. In particular, SAT4J provides an implementation of the base CDCL algorithm. JWASP extends this algorithm by modifying the propagate function of SAT4J, which in our solver includes the well founded inference rule. In particular, specific data structures and the algorithm for computing unfounded sets are introduced in JWASP which are not provided by SAT4J.

4 Experiment

The performance of JWASP was compared with CLASP 3.1.1 and LP2SAT [13]. CLASP is a native state of the art ASP solver, while LP2SAT is an ASP solver based on a rewriting of the ASP program into a SAT formula that is evaluated using a SAT solver. Two variants of LP2SAT were considered, namely LP2GLUCOSE and LP2SAT4J, which use GLUCOSE [4] and SAT4J [15] as SAT solver, respectively. All the ASP solvers use GRINGO [10] as grounder. The experiment concerns a comparison of the solvers on publicly available benchmarks used in the 3rd and 4th ASP competitions [1, 6]. The experiment was run on a four core Intel Xeon CPU X3430 2.4 GHz, with 16 GB of physical RAM, and operating system Debian Linux. Time and memory limits were set to 600 seconds and 15 GB, respectively. Performance was measured using the tools pyrunlim and pyrunner (`https://github.com/alviano/python`).

Table 1 summarizes the number of solved instances and the average running time in seconds for each solver. In particular, the first column reports the considered benchmarks; the remaining columns report the number of solved instances within the time-out (solved), and the running time averaged over solved instances (time). The first observation is that JWASP outperforms the rewriting-based LP2SAT4J. In fact, JWASP solved 17 more instances than LP2SAT4J and it is in general faster. The advantage of JWASP is obtained in 3 different benchmarks, namely KnightTour, MazeGeneration, and Numberlink, where JWASP solves 5, 7, and 3 more instances than LP2SAT4J. Once the SAT solver backhand is replaced by GLUCOSE, a clear improvement of performance is measured. LP2GLUCOSE is clearly faster (it solves 20 instances more) than LP2SAT4J. In

Table 1. Solved instances and average running time.

		LP2SAT4J		JWASP		LP2GLUCOSE		WASP		CLASP	
Track	**#**	**sol.**	**avg t**	**sol.**	**avg t**	**sol.**	**avg t**	**sol.**	**avg t**	**sol.**	**avg t**
GraphColouring	30	8	47.45	7	31.07	14	124.02	8	66.15	13	129.98
HanoiTower	30	27	120.80	26	166.57	30	10.41	30	33.83	28	53.18
KnightTour	10	2	67.66	7	52.03	3	24.37	8	4.39	10	57.95
Labyrinth	30	14	222.34	17	158.44	18	151.70	26	72.64	26	48.05
MazeGeneration	10	3	332.46	10	5.06	4	164.15	10	3.10	10	1.04
Numberlink	10	4	98.05	7	7.67	5	164.67	8	12.71	8	7.91
SokobanDecision	10	6	46.57	7	61.42	10	59.34	9	92.15	10	42.91
Total	130	64	133.72	81	100.50	84	82.45	99	44.75	105	52.48

this case, since the rewriting technique is the same, the difference of performance is due to the fact that GLUCOSE outperforms LP2SAT4J. The performance gap between C++ and Java implementations can be observed also by comparing WASP and JWASP. In particular, WASP solves 18 more instances than JWASP. The differences are noticeable in Labyrinth where WASP solves 9 more instances than JWASP. Similar considerations hold by comparing CLASP and JWASP. In fact, the former is in general faster solving 24 more instances than the latter. Finally, it is important to note that JWASP is basically comparable in performance with LP2GLUCOSE (the latter solves only 3 instances more than the former). An in-depth analysis shows that JWASP is faster in KnightTour and MazeGeneration solving 4 and 6 instances more than LP2GLUCOSE, respectively. On the contrary, LP2GLUCOSE is faster than JWASP in GraphColouring, HanoiTower, and SokobanDecision. We observe that the main advantage of JWASP over LP2GLUCOSE is registered (as expected) in the benchmarks in which the well founded propagation (implemented natively by JWASP) is applied, such as KnightTour and MazeGeneration.

5 Discussion

During recent years, ASP has obtained growing interest since efficient implementations were available. For reason of efficiency, most of the modern ASP solver are implemented in C++. To the best of our knowledge, the only previous Java-based ASP solver was JSMODELS [14], which is not developed anymore. JSMODELS was based on SMODELS featuring the DPLL algorithm and lookahead heuristics. From an abstract point of view, JWASP is more similar to modern ASP solvers, like WASP [2, 3] and CLASP [11]. In fact, all the three solvers are based on CDCL algorithm and source pointers for the computation of unfounded sets. However, JWASP is implemented in Java and thus it is a cross-platform and more portable implementation. An alternative to the development of a native solver is to rewrite the input program into a CNF formula, as done by the family of solvers LP2SAT [13]. This alternative approach can be applied to obtain a Java-based solver by endowing LP2SAT with a Java-based SAT solver such as SAT4J. This approach is less efficient than JWASP in the experimental analysis reported in this paper. It is worth noting that, both JWASP and LP2SAT apply the Clark's completion [17]. Thus, the main difference between JWASP and LP2SAT4J consists of the native computation of unfounded set of JWASP, which is obtained by using an algorithm based on source pointers introduced by SMODELS [23].

In this paper we reported on the new Java-based ASP solver JWASP built on the top of the SAT solver SAT4J. The new solver was compared with both C++ and Java-based approaches. In our experiment, JWASP outperforms the Java-based alternative LP2SAT4J, and it is competitive with LP2GLUCOSE. However, as expected, JWASP is in general slower than the native solvers. This confirms that C++ implementations are usually much faster than Java-based approaches as also noted in [15]. Future work concerns the extension of JWASP for handling optimization constructs and cautious reasoning.

References

1. M. Alviano, F. Calimeri, G. Charwat, M. Dao-Tran, C. Dodaro, G. Ianni, T. Krennwallner, M. Kronegger, J. Oetsch, A. Pfandler, J. Pührer, C. Redl, F. Ricca, P. Schneider, M. Schwen-

gerer, L. K. Spendier, J. P. Wallner, and G. Xiao. The fourth answer set programming competition: Preliminary report. In P. Cabalar and T. C. Son, editors, *LPNMR*, LNCS, pages 42–53, 2013.

2. M. Alviano, C. Dodaro, W. Faber, N. Leone, and F. Ricca. WASP: A native ASP solver based on constraint learning. In P. Cabalar and T. C. Son, editors, *LPNMR*, volume 8148 of *LNCS*, pages 54–66. Springer, 2013.
3. M. Alviano, C. Dodaro, N. Leone, and F. Ricca. Advances in WASP. In F. Calimeri, G. Ianni, and M. Truszczynski, editors, *LPNMR*, volume 9345 of *LNAI*, page (to appear), 2015.
4. G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In C. Boutilier, editor, *IJCAI*, pages 399–404, 2009.
5. G. Brewka, T. Eiter, and M. Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.
6. F. Calimeri, G. Ianni, and F. Ricca. The third open answer set programming competition. *TPLP*, 14(1):117–135, 2014.
7. C. Dodaro, B. Nardi, N. Leone, and F. Ricca. Allotment problem in travel industry: A solution based on ASP. In B. ten Cate and A. Mileo, editors, *RR*, volume 9209 of *LNCS*. Springer, 2015.
8. N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *SAT*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.
9. O. Febbraro, K. Reale, and F. Ricca. ASPIDE: integrated development environment for answer set programming. In J. P. Delgrande and W. Faber, editors, *LPNMR*, volume 6645 of *LNCS*, pages 317–330. Springer, 2011.
10. M. Gebser, R. Kaminski, A. König, and T. Schaub. Advances in *gringo* series 3. In J. P. Delgrande and W. Faber, editors, *LPNMR*, volume 6645 of *LNCS*, pages 345–351. Springer, 2011.
11. M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012.
12. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
13. 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.
14. H. V. Le and E. Pontelli. A Java Based Solver for Answer Set Programming.
15. D. Le Berre and A. Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.
16. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM TOCL*, 7(3):499–562, 2006.
17. F. Lin and Y. Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.
18. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535. ACM, 2001.
19. F. Ricca. The DLV java wrapper. In F. Buccafurri, editor, *AGP*, pages 263–274, 2003.
20. F. Ricca, A. Dimasi, G. Grasso, S. M. Ielpa, S. Iiritano, M. Manna, and N. Leone. A logic-based system for e-tourism. *Fundam. Inform.*, 105(1-2):35–55, 2010.
21. F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone. Team-building with answer set programming in the gioia-tauro seaport. *TPLP*, 12(3):361–381, 2012.
22. J. P. M. Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.
23. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234, 2002.

Searching for Sequential Plans Using Tabled Logic Programming

Roman Barták and Jindřich Vodrážka

Charles University in Prague, Faculty of Mathematics and Physics, Czech Republic

Abstract. Logic programming provides a declarative framework for modeling and solving many combinatorial problems. Until recently, it was not competitive with state of the art planning techniques partly due to search capabilities limited to backtracking. Recent development brought more advanced search techniques to logic programming such as tabling that simplifies implementation and exploitation of more sophisticated search algorithms. Together with rich modeling capabilities this progress brings tabled logic programing on a par with current best planners. The paper brings an initial experimental study comparing various approaches to search for sequential plans in the Picat planning module.

Keywords: planning; tabling; iterative deepening; branch-and-bound

1 Introduction

Automated planning was an important area for Prolog. PLANNER [5] was designed as a language for proving theorems and manipulating models in a robot, and it is perceived as the first logic programming (LP) language. Nevertheless, since the design of STRIPS planning model [6], planning approaches other than LP were more successful. SAT-based planning [9] is probably the closest approach to logic programming that is competitive with best automated planners.

For decades the so called domain-independent planning has been perceived as the major direction of AI research with the focus on "physics-only" planning domain models. This attitude is represented by International Planning Competitions (IPC) [8] that accelerated planning research by providing a set of standard benchmarks. On the other hand and despite the big progress of domain-independent planners in recent years, these planning approaches are still rarely used in practice. For example, it is hard to find any of these planners in areas such as robotics and computer games. This is partly due to low efficiency of the planners when applied to hard real-life problems and partly due to missing guidelines about how to describe planning problems in such a way that they are efficiently solvable.

IPC accelerated research in domain-independent planning by providing encodings (domain models) for many benchmark problems. On the other hand, as everyone is using IPC benchmark problems to evaluate the planners, there has not been almost any research about how to encode the planning problems efficiently. Also, though the role of domain knowledge is well known in planning [4], the domain-dependent planners were banned from IPC which further decreased interest in alternative approaches to model and solve planning problems.

Recently, tabling has been successfully used to solve specific planning problems such as Sokoban [20], the Petrobras planning problem [2], and several planning problems used in ASP competitions [23]. This led to development of the `planner` module of the Picat programming language. This general planning system was applied to various domains in IPC and compared with best domain-independent optimal planners [24] as well as best domain-dependent planners [3]. In this paper we summarize the modeling and solving capabilities of Picat and we focus on their deeper experimental comparison.

2 Background on Planning

Classical AI planning deals with finding a sequence of actions that change the world from some initial state to a goal state. We can see AI planning as the task of finding a path in a directed graph, where nodes describe states of the world and arcs correspond to state transitions via actions. Let $\gamma(s, a)$ describe the state after applying action a to state s, if a is applicable to s (otherwise the function is undefined). Then the planning task is to find a sequence of actions $\langle a_1, a_2, \ldots, a_n\rangle$ called a *plan* such that, s_0 is the initial state, for each $i \in \{1, \ldots, n\}$, a_i is applicable to the state s_{i-1} and $s_i = \gamma(s_{i-1}, a_i)$, and, finally, s_n satisfies a given goal condition. For solving cost-optimization problems, each action has assigned a non-negative cost and the task is to find a plan with the smallest cost.

As the state space is usually huge, an implicit and compact representation of states and actions is necessary. Since the time of Shakey, the robot [15, 6], a *factored representation* of states is the most widely used. Typically, the state of the world is described as a set of predicates that hold in the state or by a set of values for multi-valued state variables. Actions are then describing changes of the states in the representation, for example, actions make some predicates true and other false or actions change values of certain states variables. The Planning Domain Definition Language (PDDL) [13] is the most widely used modeling language for describing planning domains using the factored representation of states. This is also the language of IPC competitions.

In Picat we will preserve the state-transition nature of classical AI planning, but instead of factored representation we will use a *structured representation* of states. Like in the PDDL, each action will have pre-conditions verifying whether the action is applicable to a given state. However, the precondition can be any Picat call. The action itself will specify how the state should be changed; we will give some examples later.

3 Background on Picat

Picat is a logic-based multi-paradigm programming language aimed for general-purpose applications. Picat is a rule-based language, in which predicates, functions, and actors are defined with pattern-matching rules. Picat incorporates many declarative language features for better productivity of software development, including explicit non-determinism, explicit unification, functions, list comprehensions, constraints, and tabling.

In Picat, predicates and functions are defined with pattern-matching rules. Picat has two types of rules: a *non-backtrackable rule* (also called a *commitment rule*) $Head, Cond$ `=>` *Body*, and a backtrackable rule $Head, Cond$ `?=>` *Body*. In a predicate definition, the $Head$ takes the form $p(t_1, \ldots, t_n)$, where p is called the predicate name, and n is called the arity. The condition $Cond$, which is an optional goal, specifies a condition under which the rule is applicable. For a call C, if C matches $Head$ and $Cond$ succeeds, then the rule is said to be *applicable* to C. When applying a rule to call C, Picat rewrites C into *Body*. If the used rule is non-backtrackable, then the rewriting is a commitment, and the program can never backtrack to C. However, if the used rule is backtrackable, then the program will backtrack to C once *Body* fails, meaning that *Body* will be rewritten back to C, and the next applicable rule will be tried on C.

Briefly speaking, Picat programming is very similar to Prolog programming. By providing features like functions, list comprehensions etc., Picat programs are even more compact and declarative than equivalent Prolog programs. Moreover, the possibility of explicit non-determinism and unification gives the programmer better control of program execution to make the code even more efficient. More details about the Picat language can be found in the Picat documentation [16].

3.1 Tabling

The Picat system provides a built-in tabling mechanism [21] that simplifies coding of some search algorithms. Tabling is a technique to memorize answers to calls and re-using the answer when the same call appears later. Tabling implicitly prevents loops and brings properties of graph search (not exploring the same state more than once) to classical depth-first search used by Prolog-like languages. Both predicates and functions can be tabled; linear tabling [21] is used in Picat. In order to have all calls and answers of a predicate or a function tabled, users just need to add the keyword `table` before the first rule. For a predicate definition, the keyword `table` can be followed by a tuple of table modes [7], including `+` (input), `-` (output), `min`, `max`, and `nt` (not tabled). These modes specify how a particular attribute of the predicate should be handled. For a predicate with a table mode declaration that contains `min` or `max`, Picat tables one optimal answer for each tuple of the input arguments. The last mode can be `nt`, which indicates that the corresponding argument will not be tabled [22]. Ground structured terms are hash-consed [19] so that common ground terms are tabled only once. For example, for three terms `c(1,c(2,c(3)))`, `c(2,c(3))`, and `c(3)`, the shared sub-terms `c(2,c(3))` and `c(3)` are reused from `c(1,c(2,c(3)))`.

Mode-directed tabling has been successfully used to solve specific planning problems such as Sokoban [20], and the Petrobras planning problem [2]. A planning problem is modeled as a path-finding problem over an implicitly specified graph. The following code gives the framework used in all these solutions:

```
table (+,-,min)
path(S,Path,Cost), final(S) => Path = [], Cost = 0.

path(S,Path,Cost) =>
    action(S,S1,Action,ActionCost),
    path(S1,Path1,Cost1),
    Path = [Action|Path1],
    Cost = Cost1+ActionCost.
```

The call `path(S,Path,Cost)` binds `Path` to an optimal path from `S` to a final state. The predicate `final(S)` succeeds if `S` is a final state, and the predicate `action` encodes the set of actions in the problem.

3.2 Resource-Bounded Search

As mentioned in the previous section, the tabling mechanism supports solving optimization problems, such as looking for the shortest path, using the table modes `min` and `max`. When applied to the single-source shortest path problem, linear tabling is similar to Dijkstra's algorithm, except that linear tabling tables shortest paths from the encountered states to the goal state rather than shortest paths to the encountered states from the initial state. When looking for the shortest path from a single initial state to some goal state only, such as in planning, classical tabling may be too greedy as it visits the states that could be farther from the initial state than the length of the shortest path from start to goal. Resource-bounded search is a way to overcome this inefficiency.

Assume that we know the upper bound for the path length, let us call it a resource. Each time, we expand some state, we decrease available resource by the cost of the action used for expansion. Hence less quantity of resource will be available for expansion of the next state (if action costs are positive). The idea of resource-bounded search is to utilize tabled states and their resource limits to effectively decide when a state should be expanded and when a state should fail. Let S^R denote a state with an associated resource limit, R. If R is negative, then S^R immediately fails. If R is non-negative and S has never been encountered before, then S is expanded by using a selected action. Otherwise, if the same state S has failed before and R' was the resource limit when it failed, then S^R is only expanded if $R > R'$, i.e., if the current resource limit is larger than the resource limit was at the time of failure.

4 Planning in Picat

The Picat system has a built-in module `planner` for solving planning problems. The planning problem is described as an abstract state transition diagram and solved using techniques exploiting tabling. By abstraction we mean that states and actions are not grounded, but described in an abstract way similar to modeling operators in PDDL. In this section we briefly introduce the `planner` module, give an example of planning domain model in Picat, and describe available search techniques to solve the planning problems.

4.1 The `planner` Module of Picat

The `planner` module is based on tabling but it abstracts away tabling from users. For a planning problem, users only need to define the predicates `final/1` and `action/4`, and call one of the search predicates in the module on an initial state in order to find a plan or an optimal plan.

– `final(`S`)`: This predicate succeeds if S is a final state.
– `action(`S`,`$NextS$`,`$Action$`,`$ACost$`)`: This predicate encodes the state transition diagram of a planning problem. The state S can be transformed to $NextS$ by performing *Action*. The cost of *Action* is $ACost$, which must be non-negative. If the plan's length is the only interest, then $ACost = 1$.

These two predicates are called by the planner. The `action` predicate specifies the precondition, effect, and cost of each of the actions. This predicate is normally defined with nondeterministic pattern-matching rules. As in Prolog, the planner tries actions in the order they are specified. When a non-backtrackable rule is applied to a call, the remaining rules will be discarded for the call.

4.2 Modeling Example

To demonstrate how the planning domain is encoded in Picat, we will use the *Transport* domain from IPC'14. Given a weighted directed graph, a set of trucks each of which has a capacity for the number of packages it can carry, and a set of packages each of which has an initial location and a destination, the objective of the problem is to find an optimal plan to transport the packages from their initial locations to their destinations. This problem is more challenging than the *Nomystery* problem that was used in IPC'11, because of the existence of multiple trucks, and because an optimal plan normally requires trucks to cooperate. This problem degenerates into the shortest path problem if there is only one truck and only one package. We introduced the Picat model of this domain in [24], where other examples of domain models are given.

A state is represented by an array of the form {`Trucks,Packages`}, where `Trucks` is an ordered list of trucks, and `Packages` is an ordered list of waiting packages. A package in `Packages` is a pair of the form `(Loc,Dest)` where `Loc` is the source location and `Dest` is the destination of the package. A truck in `Trucks` is a list of the form `[Loc,Dests,Cap]`, where `Loc` is the current location of the truck, `Dests` is an ordered list of destinations of the loaded packages on the truck, and `Cap` is the capacity of the truck. At any time, the number of loaded packages must not exceed the capacity.

Note that keeping `Cap` as the last element of the list facilitates sharing, since the suffix `[Cap]`, which is common to all the trucks that have the same capacity, is tabled only once. Also note that the names of the trucks and the names of packages are not included in the representation. Two packages in the waiting list that have the same source and the same destination are indistinguishable, and as are two packages loaded on the same truck that have the same destination. This representation breaks object symmetries – two configurations that only differ by a truck's name or a package's name are treated as the same state.

A state is final if all of the packages have been transported.

```
final({Trucks,[]}) =>
    foreach([_Loc,Dests|_] in Trucks)
        Dests == []
    end.
```

The PDDL rules for the actions are straightforwardly translated into Picat as follows.

```
action({Trucks,Packages},NextState,Action,ACost) ?=>
    Action = $load(Loc), ACost = 1,
    select([Loc,Dests,Cap],Trucks,TrucksR),
    length(Dests) < Cap,
    select((Loc,Dest),Packages,PackagesR),
    NewDests = insert_ordered(Dests,Dest),
    NewTrucks = insert_ordered(TrucksR,[Loc,NewDests,Cap]),
    NextState = {NewTrucks,PackagesR},
action({Trucks,Packages},NextState,Action,ACost) ?=>
    Action = $unload(Loc), ACost = 1,
    select([Loc,Dests,Cap],Trucks,TrucksR),
    select(Dest,Dests,DestsR),
    NewTrucks = insert_ordered(TrucksR,[Loc,DestsR,Cap]),
    NewPackages = insert_ordered(Packages,(Loc,Dest)),
    NextState = {NewTrucks,NewPackages}.
action({Trucks,Packages},NextState,Action,ACost) =>
    Action = $move(Loc,NextLoc),
    select([Loc|Tail],Trucks,TrucksR),
    road(Loc,NextLoc,ACost),
    NewTrucks = insert_ordered(TrucksR,[NextLoc|Tail]),
    NextState = {NewTrucks,Packages}.
```

For the *load* action, the rule nondeterministically selects a truck that still has room for another package, and nondeterministically selects a package that has the same location as the truck. After loading the package to the truck, the rule inserts the package's destination into the list of loaded packages of the truck. Note that the rule is nondeterministic. Even if a truck passes by a location that has a waiting package, the truck may not pick it. If this rule is made deterministic, then the optimality of plans is no longer guaranteed, unless there is only one truck and the truck's capacity is infinite.

The above model is very similar to the PDDL encoding available at IPC web pages [8]. The major difference is the model of states that is a structure consisting of two ordered lists. The ordering is used to obtain a unique representation of states. The encoding can be further extended by adding control knowledge, for example the predicate `action` can begin with a rule that deterministically unloads a package if the package's destination is the same as the truck's location. To exploit better the resource-bound search, one can also add heuristics to action definition. The heuristic can estimate the cost-to-goal and it can be added to actions through the following condition:

```
    current_resource() - ACost >=  estimated_cost(NewState).
```

The `current_resource()` is a built-in function of the planner giving the maximal allowed cost-distance to the goal. Note that heuristic is a part of the domain model so it is domain dependent.

We discussed some domain modeling principles in [3]. Basically, the Picat `planner` module supports:

- *structured state representation* that is more compact than the factored representation and allows removing symmetry between objects by representing objects via their properties rather than via their names (see representation of trucks and packages in the *Transport* domain),
- *control knowledge* that guides the planner via ordering of actions in the model and using extra conditions to specify when actions are applicable (for example, always unload the package when the truck is at the package destination),
- *action symmetry breaking* by modeling possible action sequences via a non-deterministic finite state automaton (for example, load the truck and move it somewhere for further loading or unloading before assuming actions of another truck),
- *heuristics* that estimate the cost-to-goal and can be domain dependent (domain independent heuristics can be used as well).

4.3 Search Techniques

The planning-domain model is specified as a set of Picat rules that are explored by the Picat planner. This planner uses basically two search approaches to find optimal plans. Both of them are based on depth-first search with tabling and in some sense they correspond to classical forward planning. It means that they start in the initial state, select an action rule that is applicable to the current state, apply the rule to generate the next state, and continue until they find a state satisfying the goal condition (or the resource limit is exceeded).

The first approach starts with finding any plan using the depth first search. The initial limit for plan cost can (optionally) be imposed. Then the planner tries to find a plan with smaller cost so a stricter cost limit is imposed. This process is repeated until no plan is found so the last plan found is an optimal plan. This approach is very close to *branch-and-bound* technique [12]. Note that tabling is used there – the underlying solver remembers the best plans found for all visited states so when visiting the state next time, the plan from it can be reused rather than looked for again. This planning algorithm is evoked using the following call:

```
    best_plan_bb(+InitState,+CostLimit,-Plan,-PlanCost)
```

This is where the user specifies the initial state and (optionally) the initial cost limit. The algorithm returns a cost-optimal plan and its cost. This approach can be also used to find the first plan using the call `plan(+S,+L,-P,-C)`.

Despite using tabling that prevents re-opening the same state, this approach still requires good control knowledge to find the initial plan (otherwise, it may be lost in a huge state space) or alternatively some good initial cost limit should be used to prevent exploring long plans.

The second approach exploits the idea of iteratively extending the plan length as proposed first for SAT-based planners [9]. It first tries to find a plan with cost zero. If no plan is found, then it increases the cost by 1. In this way, the first plan that is found is guaranteed to be optimal. Unlike the *IDA* search algorithm* [10], which starts a new round from scratch, Picat reuses the states that were tabled in the previous rounds. This planning algorithm is evoked using the call:

```
best_plan(+InitState,+CostLimit,-Plan,-PlanCost)
```

This approach is more robust with respect to weak or no control knowledge, but it has the disadvantage that it can only find the optimal plan, which could be more time consuming than finding any plan.

Note that the cost limit in the above calls is used to define the function `current_resource()` mentioned in the action rules. Briefly speaking the cost of the partial plan is subtracted from the cost limit to get the value of the function `current_resource()` that can be utilized to compare with the heuristic distance to the goal.

5 Experimental Comparison

The Picat planner uses a different approach to planning so it is important to show how this approach compares with current state-of-the-art planning techniques and to understand better the Picat search procedures. In [24] we compared the Picat planer with SymBA [18] – the domain-independent bidirectional A* planner which won the optimal sequential track of IPC'14. As the Picat planner can exploit domain-dependent information, in [3] we compared the Picat planner with leading domain-dependent planners based on control rules and hierarchical task networks (HTN). We will summarize these results first and then we will present a new experimental study comparing the search approaches in Picat.

5.1 Comparison to Automated Planners

Optimal Domain Independent Planners. We have encoded in Picat most domains used in the deterministic sequential track of IPC'14. All of the encodings are available at: `picat-lang.org/ipc14/`. The Picat planner was using the iterative deepening `best_plan/4` planning algorithm. We have compared these Picat encodings with the IPC'14 PDDL encodings solved with SymBA. Table 1 shows the number of instances (#insts) in the domains used in IPC'14 and the number of (optimally) solved instances by each planner. The results were obtained on a Cygwin notebook computer with 2.4GHz Intel i5 and 4GB RAM. Both Picat and SymBA were compiled using g++ version 4.8.3. For SymBA, a setting suggested by one of SymBA's developers was used. A time limit of 30 minutes was used for each instance as in IPC. For every instance solved by both planners, the plan quality is the same. The running times of the instances are not given, but the total runs for Picat were finished within 24 hours, while the total runs for SymBA took more than 72 hours.

Table 1. The number of problems solved optimally.

Domain	# insts	Picat	SymBA
Barman	14	**14**	6
Cave Diving	20	**20**	3
Childsnack	20	**20**	3
Citycar	20	**20**	17
Floortile	20	**20**	**20**
GED	20	**20**	19
Parking	20	**11**	1
Tetris	17	**13**	10
Transport	20	**10**	8
Total	171	**148**	87

Domain Dependent Planners. We took the following domains: *Depots*, *Zenotravel*, *Driverlog*, *Satellite*, and *Rovers* from IPC'02. The Picat encodings are available at: `picat-lang.org/aips02/`. We compared Picat with TLPlan [1], the best hand-coded planner of IPC'02, TALPlanner [11] another good planner based on control rules, and SHOP2 [14], the distinguished hand-coded planner of IPC'02 using HTN. Each of these planners used its own encoding of planning domains developed by the authors of the planners.

All planners found (possibly sub-optimal) plans for all benchmark problems and the runtime to generate plans was negligible; every planner found a plan in a matter of milliseconds. Hence we focused on comparing the quality of obtained plans that is measured by a so called *quality score* introduced in IPC. Briefly speaking the score for solving one problem is 1, if the planner finds the best plan among all planners; otherwise the score goes down proportionally to the quality of the best plan found. The higher quality score means an overall better system.

For TLPlan, TALPlanner, and SHOP2 we took the best plans reported in the results of IPC'02. Taking in account the nature of planners and their runtimes, there is a little hope to get better plans when running on the current hardware. For the Picat planner we used the branch-and-bound `best_plan_bb/4` planning algorithm. Table 2 shows the quality scores when we gave five minutes to the Picat planner to improve the plan (running under MacOS X 10.10 on 1.7 GHz Intel Core i7 with 8 GB RAM).

The results show that the Picat planner is competitive with other domain-dependent planners and that it can even find better plans. In [3] we also demon-

Table 2. Comparison of quality scores for the best plan (5 minutes)

Domain	# insts	Picat	TLPlan	TALPlanner	SHOP2
Depots	22	**21.94**	19.93	20.52	18.63
Zenotravel	20	**19.86**	18.40	18.79	17.14
Driverlog	20	17.21	17.68	**17.87**	14.16
Satellite	20	**20.00**	18.33	16.58	17.16
Rovers	20	**20.00**	17.67	14.61	17.57
Total	102	**99.01**	92.00	88.37	84.65

strated that the Picat domain models are much smaller than domain models using control rules and are much closer in size to the PDDL models.

5.2 Comparison of Search Techniques

In the second experiment we focused on comparing two search approaches to find cost-optimal plans in Picat, namely branch-and-bound and iterative deepening. When looking for optimal plans, the hypothesis is that iterative deepening requires less memory and time because branch-and-bound explores longer plans and hence may visit more states. On the other hand, the advantage of branch-and-bound is that it can find some plan even if finding (and proving) optimal plan is hard (recall, that iterative deepening returns only optimal plans). So the second hypothesis is that when looking for any plan, branch-and-bound could be a better planning approach. Nevertheless, due to depth-first-search nature, branch-and-bound requires good control knowledge to find an initial plan. The final hypothesis is that if none or weak control knowledge is part of the domain model then iterative deepening is a more reliable planning approach.

We used the following domains from the deterministic sequential track of IPC'14 [8]: *Barman*, *Cavediving*, *Childsnack*, *Citycar*, *Floortile*, *GED*, *Parking*, *Tetris*, and *Transport*. All of the encodings are available at: `picat-lang.org/ipc14/`. The experiment run on Intel Core i5 (Broadwell) 5300U(2.3/2.9GHz) with 4 GB RAM (DDR3 1600 MHz). For each problem, we used timeout of 30 minutes and memory limit 1 GB. We compared the following search procedures:

- `plan(`$InitState$`,`$CostLimit$`,`$Plan$`,`$PlanCost$`)`,
- `best_plan(`$InitState$`,`$CostLimit$`,`$Plan$`,`$PlanCost$`)`,
- `best_plan_bb(`$InitState$`,`$CostLimit$`,`$Plan$`,`$PlanCost$`)`,

using $99,999,999$ as the initial cost limit ($10,000$ for the *GED* domain).

We first report the number of solved problems with respect to time and memory consumed. Note that `best_plan/4` and `best_plan_bb/4` return cost-optimal plans while `plan/4` returns some (possibly sub-optimal) plan. Figure 1 shows the number of solved problems within a given time. Figure 2 shows the number of solved problems based on memory consumed.

The results confirm the first and second hypotheses, that is, iterative deepening requires less time and less memory than branch-and-bound when solving

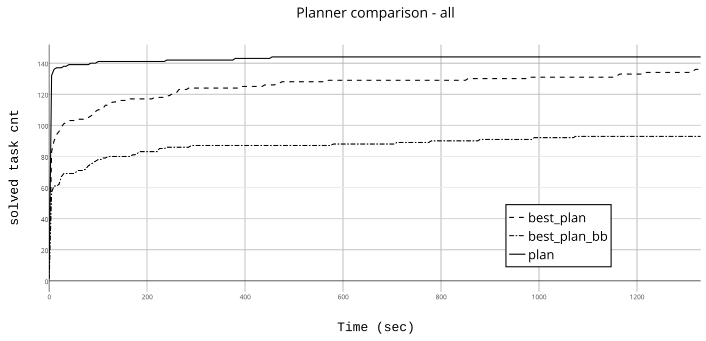


Fig. 1. The number of solved problems within a given time.

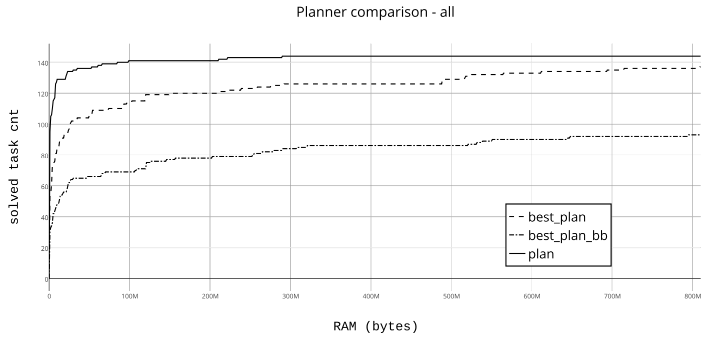


Fig. 2. The number of solved problems dependent on memory consumption.

problems optimally, but branch-and-bound has the advantage of providing some (possibly sub-optimal) plan fast. If looking for any plan then branch-and-bound also requires less memory.

Describing dependence of planner efficiency on the model is more tricky as it is hard to measure model quality quantitatively. We annotated each involved domain model by information about using control knowledge and domain-dependent heuristics in the model. Table 3 shows the annotation of domain models based on these two criteria.

Based on Table 3 we can classify the Picat domain models into following groups:

- The Picat domain model for *Barman* is probably closest to the PDDL encoding; it only uses the structured representation of states, which alone seems to be advantage over PDDL as Table 1 shows. *GED* uses a bit specific model based on a PDDL model different from that one used in the IPC – this model uses some macro-actions – and hence it is not really tuned for Picat.

Table 3. The properties of domain models.

Domain	control knowledge	heuristics
Barman	no	no
Cave Diving	strong	no
Childsnack	strong	no
Citycar	no	yes
Floortile	strong	no
GED	macro	yes
Parking	weak	yes
Tetris	no	yes
Transport	weak	yes

- *Citycar* and *Tetris* are domains where useful admissible heuristics are used, but no control knowledge is implemented to guide the planner.
- The Picat domain models for *Parking* and *Transport* use some weak control knowledge in the form of making selection of some actions deterministic (see the example earlier in the paper). They also exploit admissible heuristics.
- *Cave Diving*, *Childsnack*, and *Floortile* are domains, where we use strong control knowledge and no heuristics. Control knowledge is used there to describe reasonable sequencing of actions either via finite state automata or macro-actions. The domain model for *Cave Diving* is described in detail in [3]; the domain model for *Childsnack* is almost deterministic as this problem does not require real planning; and the domain model for *Floortile* uses macro-actions to force reasonable action sequences, see [24] for details.

From each class of domain models we selected one representative to demonstrate how different solving approaches behave (the other domains gave similar results). Figure 3 shows the number of solved problems for these representatives. If the Picat domain model is very close to the original PDDL model, then iterative deepening has a clear advantage when finding optimal plans, see the *Barman* domain. This corresponds to popularity of this solving approach in planners based on SAT techniques [9]. In case of *Barman* the branch-and-bound approach can still find some plans as the model itself guides the planner reasonably well (there are no extremely long plans). However, for the *GED* domain, only iterative deepening can find (optimal) plans while branch-and-bound was not able to find any plan due to being lost in generating extremely long plans not leading to the goal.

Adding admissible heuristics makes iterative deepening even more successful, see the *Tetris* domain. Finding optimal plans by iterative deepening is close to finding any plan by branch-and-bound. Also the gap between finding any plan and finding an optimal plan by branch-and-bound is narrower there. Obviously, this also depends on the quality of first plan found.

An interesting though not surprising observation is that adding even weak control knowledge makes finding any plan by branch-and-bound much more successful and decreases further the gap between iterative deepening and branch-

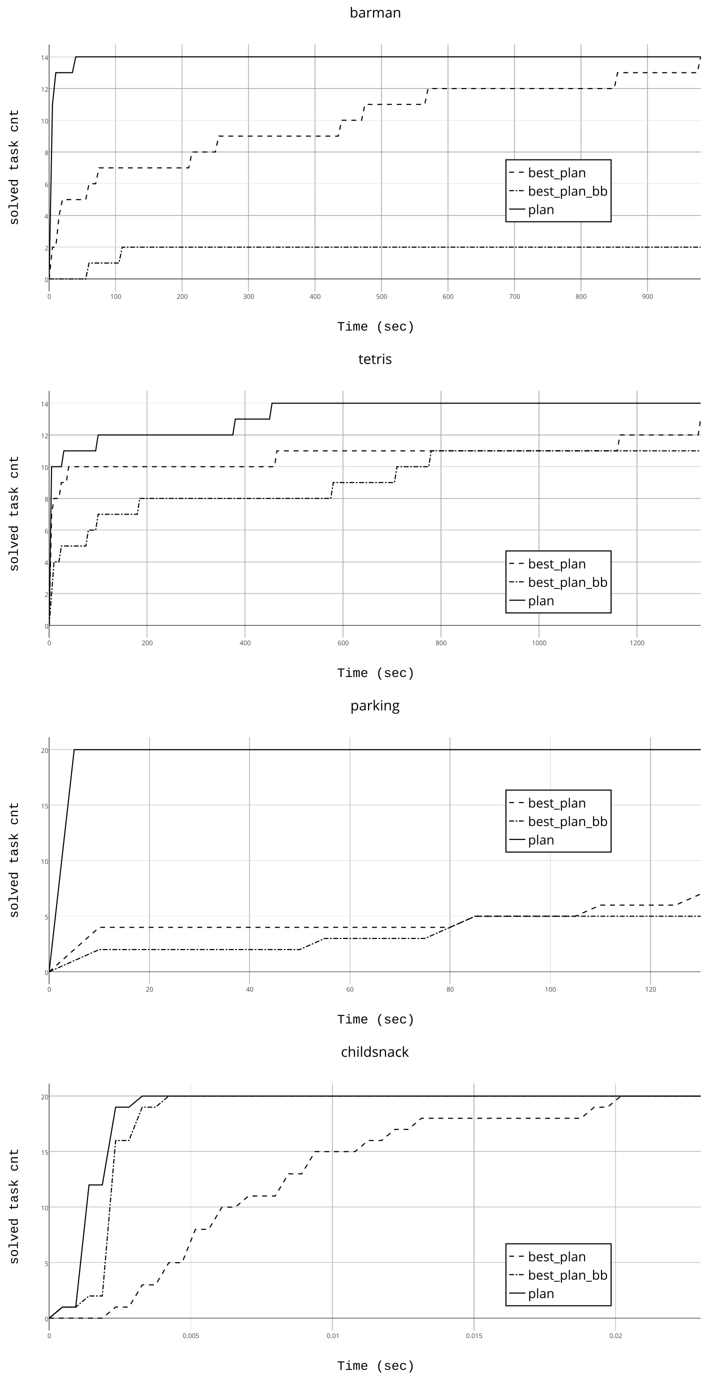


Fig. 3. The number of solved problems within a given time for specific domains.

and-bound when looking for optimal plans, see the *Parking* domain. The role of control knowledge is even more highlighted in the *Childsnack* domain, which shows that strong control knowledge has a big influence on efficiency of branch-and-bound. Longer runtimes of iterative deepening are caused by exploring short plans that cannot solve the problem before discovering the necessary length of the plan to reach the goal. Still control knowledge helps iterative deepening to find a larger number of optimal plans though it takes longer than for branch-and-bound.

The experimental results justify the role of control knowledge for solving planning problems and confirm the last hypothesis that control knowledge is important for the branch-and-bound approach especially if the dead-ends can be discovered only in very long plans.

6 Summary

This paper puts in contrast two approaches for searching for sequential plans, iterative deepening used in [24] and branch-and-bound used in [3]. We demonstrated that the modeling framework proposed for the Picat `planner` module is competitive with state-of-the-art planning approaches and we showed some relations between the modeling techniques and used search algorithms. In particular, we demonstrated the role of control knowledge in planning and we showed that control knowledge is more important for branch-and-bound though it also contributes to efficiency of iterative deepening. The role of heuristics is known in planning as for a long time heuristic-based forward planners are the leading academic planners. Note however that Picat is using heuristics in a different way. Rather than guiding the planner to promising areas of the search space, the heuristics are used to cut-off sub-optimal plans earlier. Hence the role of heuristics is stronger for iterative deepening than for branch-and-bound.

This paper showed some preliminary results on the relations between various modeling and solving techniques for planning problems. The next step is a deeper study of influence of various modeling techniques on efficiency of planning.

Acknowledgments. Research was supported by the Czech Science Foundation under the project P103-15-19877S. The authors would like to thank Agostino Dovier and Neng-Fa Zhou for providing some of the domain models in Picat.

References

1. Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.
2. Roman Barták and Neng-Fa Zhou. Using tabled logic programming to solve the Petrobras planning problem. *Theory and Practice of Logic Programming*, 14(4-5):697–710, 2014.
3. Roman Barták, Agostino Dovier, Neng-Fa Zhou. On modeling planning problems in tabled logic programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming – PPDP'15*, 32–42, 2015.

4. Patrik Haslum and Ulrich Scholz. Domain knowledge in planning: Representation and use. In *ICAPS Workshop on PDDL*, 2003.
5. Carl Hewitt. Planner: A language for proving theorems in robots. In *Proceedings of IJCAI*, 295–302, 1969.
6. Richard E. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2 (3-4):189–208, 1971
7. Hai-Feng Guo and Gopal Gupta. Simplifying dynamic programming via mode-directed tabling. *Software: Practice and Experience*, 38(1):75–94, 2008.
8. International Planning Competitions web site, `http://ipc.icaps-conference.org/`, Accessed April 5, 2015.
9. Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of ECAI*, 359–363, 1992.
10. Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
11. Jonas Kvarnström and Martin Magnusson. Talplanner in the third international planning competition: Extensions and control rules. *J. Artificial Intelligence Research* (JAIR), 20:343–377, 2003.
12. A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica* 28(3):497–520, 1960.
13. Drew McDermott. The planning domain definition language manual. CVC Report 98-003, Yale Computer Science Report 1165, 1998.
14. Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: an HTN planning system. *J. Artificial Intelligence Research* (JAIR), 20:379–404, 2003.
15. Nils J. Nilsson. Shakey The Robot, Technical Note 323. AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Apr 1984.
16. Picat web site, `http://picat-lang.org/`, Accessed July 3, 2015.
17. TLPlan web site, `http://www.cs.toronto.edu/tlplan/`, Accessed April 5, 2015.
18. Alvaro Torralba, Vidal Alcazar, and Daniel Borrajo. Symba: A symbolic bidirectional a planner. In *The 2014 International Planning Competition*,105–109, 2014.
19. Neng-Fa Zhou and Christian Theil Have. Efficient tabling of structured data with enhanced hash-consing. *Theory and Practice of Logic Programming*, 12(4-5):547–563, 2012.
20. Neng-Fa Zhou and Agostino Dovier. A tabled Prolog program for solving Sokoban. *Fundamenta Informaticae*, 124(4):561–575, 2013.
21. Neng-Fa Zhou, T. Sato, and Y.-D. Shen. Linear tabling strategies and optimizations. *Theory and Practice of Logic Programming*, 8(1):81–109, 2008.
22. Neng-Fa Zhou, Y. Kameya, and T. Sato. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *Proceedings of 22nd IEEE International Conference on Tools with Artificial Intelligence* (ICTAI), 213–218, 2010.
23. Neng-Fa Zhou. Combinatorial Search With Picat. *http://arxiv.org/abs/1405.2538*, 2014.
24. Neng-Fa Zhou, Roman Barták, Agostino Dovier. Planning as Tabled Logic Programming. To appear in *Theory and Practice of Logic Programming*, 2015.

Testing Credulous and Sceptical Acceptance in Small-World Networks

Stefano Bistarelli, Fabio Rossi, Francesco Santini

Dipartimento di Matematica e Informatica, Università di Perugia
[bista,rossi,francesco.santini]@dmi.unipg.it

Abstract. In this paper we test how efficiently state-of-the art solvers are capable of solving credulous and sceptical argument-acceptance for lower-order extensions. As our benchmark we consider two different random graph-models to obtain random Abstract Argumentation Frameworks with small-world characteristics: Kleinberg and Watt-Strogatz. We test two reasoners, i.e., ConArg2 and dynPARTIX, on such benchmark, by comparing their performance on NP/co-NP-complete decision problems related to argument acceptance in admissible, complete, and stable semantics.

1 Introduction

An *Abstract Argumentation Framework* (*AAF*) [8], or System, is simply a pair $\langle A, R \rangle$ consisting of a set A of arguments, and of a binary relation R on A, called the "attack" relation. An abstract argument is not assumed to have any specific structure but, roughly speaking, an argument is anything that may attack or be attacked by another argument. Two main styles of argumentation semantics definition can be identified in the literature: *extension-based* and *labelling-based.* In this work we exploit the extension-based approach, where a given semantics definition (related to varying degrees of scepticism or credulousness) specifies how to derive a set of extensions from an AAF, which basically consist in conflict-free subsets of A with different properties.

In this paper we are interested in the acceptance (or justification) state of arguments: intuitively an argument is regarded as accepted if it is at least once (credulously) or always (sceptically) present in all the extensions satisfying a given semantics. The kind of semantics (from le least to the most binding), and its acceptance state (e.g., credulous or sceptical) point to a strength evaluation of an argument.

In this work we move along the line activated in our previous works [5, 2, 4, 3]. Differently from these papers, here we extend [3] by considering networks with small-world topologies [15, 18]: in [3] we present a benchmark assembled with random trees, scale-free networks [1], and just random graphs [13]. The motivation is to study topologies possibly shown by advanced social debating platforms, as *DebateGraph*[1], which allow a discussion to be less rigidly structured

[1] http://debategraph.org

than a tree, as instead commonly offered in today's digital fora. Hence, in this paper we consider Kleinberg [15] and Watts-Strogatz [18] topologies.

The problems we tackle in this work correspond to the credulous acceptance in admissible, complete, and stable semantics (all NP-complete problems), and the sceptical acceptance in the stable semantics (a coNP-complete problem). We test two different solvers ConArg2 [7] and dynPARTIX [10], in order to have a comparison between them and a more informative analysis on the most efficient relation "technology against AAF topology" (i.e., Constraint Programming against Dynamic Programming).

Note that in this work we do not consider higher-order semantics, e.g., preferred or grounded [8], which can be defined from lower-order ones by selecting only the maximal or minimal ones w.r.t. set inclusion; for instance, preferred extensions are the maximal (w.r.t. set inclusion) admissible extensions. We leave their testing to future work (see Sec. 5).

2 Preliminaries

In this section we focus on the basic definitions of an AAF, and on the extension-based semantics that will be tested in our comparison (see Sec. 4).

Definition 1 (Abstract AFs). *An Abstract Argumentation Framework (AAF) is a pair $F = \langle A, R \rangle$ of a set A of arguments and a binary relation $R \subseteq A \times A$, called the attack relation. $\forall a, b \in A$, aRb (or, $a \rightarrowtail b$) means that a attacks b. An AAF may be represented by a directed graph (an interaction graph) whose nodes are arguments and edges represent the attack relation. A set of arguments $S \subseteq A$ attacks an argument a, i.e., $S \rightarrowtail a$, if a is attacked by an argument of S, i.e., $\exists b \in S.b \rightarrowtail a$.*

The following notion of defence [8] is fundamental to AAFs.

Definition 2 (Defence). *Given an AAF, $F = \langle A, R \rangle$, an argument $a \in A$ is defended (in F) by a set $S \subseteq A$ if for each $b \in A$, such that $b \rightarrowtail a$, also $S \rightarrowtail b$ holds. Moreover, for $S \subseteq A$, we denote by S_R^+ the set $S \cup \{b \mid S \rightarrowtail b\}$.*

The "acceptability" of an argument [8], defined under different semantics, depends on the frequency of its membership to some argument subsets, called *extensions*: such semantics characterise a collective "acceptability". In Def. 3 we report only the semantics of interest in this study.

Definition 3. *Let $F = \langle A, R \rangle$ be an AAF. A set $S \subseteq A$ is conflict-free (in F), denoted $S \in cf(F)$, iff there are no $a, b \in S$, such that $(a, b), (b, a) \in R$. For $S \in cf(F)$, it holds that*

- *$S \in adm(F)$, if each $a \in S$ is defended by S;*
- *$S \in com(F)$, if $S \in adm(F)$ and for each $a \in A$ defended by S, $a \in S$ holds;*
- *$S \in stb(F)$, if for each $a \in A \backslash S$, $S \rightarrowtail a$, i.e., $S_R^+ = A$;*

Table 1: Some known complexity results for the credulous and sceptical acceptance [16, Ch. 5]: in bold, the problems we test in this study.

	adm	com	stb
Credulous acc.	**NP-c**	**NP-c**	**NP-c**
Sceptical acc.	trivial	P-c	**coNP-c**

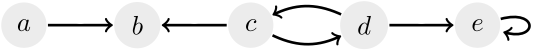


Fig. 1: A graphical example of an AAF.

We recall that for each AAF, $stb(F) \subseteq com(F) \subseteq adm(F)$ holds, and that $adm(F) \neq \emptyset$, and $com(F) \neq \emptyset$ always hold, while $stb(F) = \emptyset$ may happen instead.

Definition 4 (Acceptance state). *Given a semantics σ (e.g.,* stb*) and a framework F, an argument a is* i) *sceptically accepted iff $\forall E \in \sigma(F), a \in E$, and* ii) *credulously accepted if $\exists E \in \sigma(F), a \in E$.*

Checking the credulous/sceptical acceptance of an argument is sometimes a (time) complex problem (e.g., with the grounded semantics they are polynomial): in Tab. 1 we report in bold the complexity class of the problems we tackle in this paper, i.e., the credulous acceptance in the admissible, complete, and stable semantics (all NP-complete problems), and the sceptical acceptance in the stable semantics (a coNP-complete problem).

Consider the $F = \langle A, R \rangle$ in Fig. 1, with $A = \{a, b, c, d, e\}$ and $R = \{(a, b), (c, b), (c, d), (d, c), (d, e), (e, e)\}$. We have that $stb(F) = \{\{a, d\}\}$. The admissible extensions of F are $adm(F) = \{\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$, while the complete ones are $com(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$. For instance, argument a is both credulously and sceptically accepted in $stb(F)$ and $com(F)$, while it is only credulously accepted in $adm(F)$.

3 Tools and Graphs

In this section we introduce how we performed our tests, by describing the analysed tools (Sec. 3.1) and the generated AAFs (Sec. 3.2).

3.1 Tools

dynPARTIX[2] is a system based on decomposition and dynamic programming; it is motivated by the theoretical results in [10]. The underneath algorithms make use of the graph-parameter tree-width, which measures the "tree-likeness" of a graph. More specifically, tree-width is defined via the so-called

[2] `http://www.dbai.tuwien.ac.at/proj/argumentation/dynpartix/`

tree-decompositions. A tree decomposition is a mapping from an AAF to a tree where the nodes in the tree contain *bags* of arguments from the AAF. Each argument appears in at least one bag, adjacent arguments are together in at least one bag, and bags containing the same argument are connected. The benchmarks in [9] show that the run-time performance of dynPARTIX heavily depends on the tree-width of the considered graph: for example, with instances of small tree-width, dynPARTIX outperforms ASPARTIX [12], while with a high tree-width ASPARTIX still performs comparably (better on credulous acceptance, still worse on sceptical acceptance). In our tests we use the new 64-bit version (2.0) of dynPARTIX, which has recently become available.

ConArg2. ConArg[3] [7] is our solver based on the *Java Constraint Programming* solver[4] (JaCoP), a Java library that provides a *Finite Domain Constraint Programming* paradigm [17]. The tool comes with a graphical interface, which visually shows all the obtained extensions for each problem. ConArg is able to solve also the weighted and coalition-based problems presented in [6]. Moreover, it can import/export AAFs with the same text format of ASPARTIX. Recently, we have extended the tool to its second version, i.e., ConArg2 (freely downloadable from the same Web-page of ConArg), in order to improve its performance: we implemented all the models in Gecode[5], which is an open, free, and efficient C++ environment where to develop constraint-based applications. Hence, we model each semantics as a *Constraint Satisfaction Problem* (*CSP*) [17]. We have also dropped the graphical interface, having a textual output only, with the purpose to have a tool exclusively oriented to performance. So far, on classical AAFs, ConArg2 is able to find all conflict-free, admissible, complete, stable, grounded, preferred, semi-stable, and ideal extensions; moreover, it solves the credulous and sceptical acceptance of arguments given the admissible, complete, and stable semantics, and the existence of a stable extension (which is an NP-complete problem).

3.2 Graphs

The justification behind using Kleinberg and Watts-Strogatz models is that several works in the Argumentation literature investigate AAFs extracted from social networks [14]. However, benchmarks collected with such tools are still not available.

To generate random graphs we adopted two different libraries. The first one is the *Java Universal Network/Graph Framework* (*JUNG*[6]), which is a Java software library for the modelling, generation, analysis and visualization of graphs. With JUNG we generate Kleinberg [15] graphs. The second library we use is *NetworkX*[7], and it consists of a Python software package for the creation, manipu-

[3] `http://www.dmi.unipg.it/conarg/`

[4] `http://www.jacop.eu`

[5] `http://www.gecode.org`

[6] `http://jung.sourceforge.net`

[7] `http://networkx.github.io`

Model/Nodes	Edges	Shortest Path	Clustering C.	Diam.	InDeg.	Cycles	Min-Max Deg.
KL/16	47	1.6	0.3	2.9	5	Yes	5-11
KL/25	74	1.9	0.19	3	5	Yes	5-11
KL/36	107	2.2	0.14	3.9	5	Yes	5-11
KL/49	146	2.4	0.11	4	5	Yes	5-11
KL/64	191	2.57	0.8	4.2	5	Yes	5-12
KL/81	242	2.7	0.07	4.8	5	Yes	5-11
WS/25	50	2.4	0.22	4.7	2	Yes	2-8
WS/50	100	3.4	0.28	6.7	2	Yes	2-8
WS/80	240	2.6	0.09	4.9	3	Yes	3-13
WS/100	400	2.4	0.12	4	4	Yes	4-15

Table 2: Analysis of the generated random-AFs: values are averaged over the generated 100 AFs in each class. The considered models are Kleinberg (KL) and Watts-Strogatz (WS).

lation, and study of the structure, dynamics, and functions of complex networks. With NetworkX we generate Watts-Strogatz [18] graphs.

The **Kleinberg** [15] graph-model adds a number of directed long-range random links to an $n \times n$ lattice network. Links have a non-uniform distribution that favours edges to close nodes over more distant ones (in the number of hops). In the implementation provided by JUNG, each node u has four local connections, one to each of its neighbours, and in addition, one or more long-range connections to some node v, where v is randomly chosen according to probability proportional to $d^{-\theta}$ where d is the lattice distance between u and v and θ is the clustering exponent. In our generation we set $\theta = 0.9$, in order to have a high clustering coefficient. Given a node, each link is directed towards its neighbours: edges are created in both directions between neighbours. Each long-distance edge has the tail in the considered node.

The **Watts-Strogatz** model [18] consists in a ring over n nodes. each node in the ring is connected with its k nearest neighbours ($k-1$ neighbours if k is odd). Then shortcuts are created by replacing some edges as follows: for each edge (u, v) in the underlying "n-ring with k nearest neighbours with probability p replace it with a new edge (u, w) with uniformly random choice of existing node w. Varying p makes it possible to interpolate between a regular lattice ($p = 0$) and an Erdős-Rényi graph ($p = 1$). In our graph generation we set $k = (n/10) - 2$ and $p = 0.1$: in this way we obtain a high clustering coefficient (the max is with $k = n/2$, i.e., a complete graph) without increasing the number of edges (attacks) too much; we also obtain a graph structure closer to a lattice (p is low). Note that, since NetworkX generates undirected Watts-Strogatz graphs, we orient each edge in one of the two directions (0.5 of probability each).

For each set of 100 networks we also collected the following parameters, shown in Tab. 2: the Average Number of Edges, the Average Shortest Path (between all the couples of nodes), the Average Clustering Coefficient (the fraction of all the possible triangles through a node), the Average Diameter, the Average InDegree, the presence of Simple Cycles (closed paths where no node appears twice, except that the first and last node are the same), and the Max and Min degree for a node over the whole class.

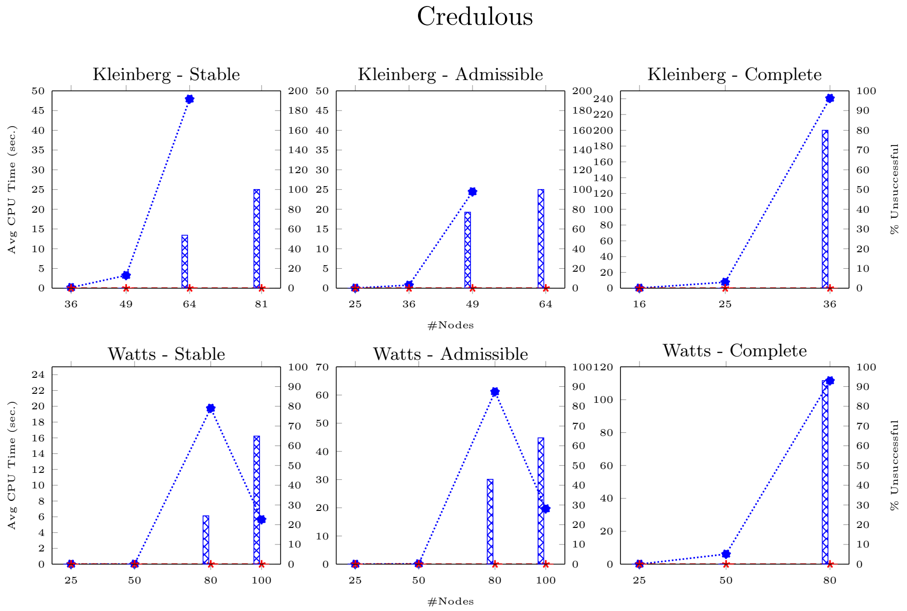


Fig. 2: Avg. time dynPARTIX (⋯◆⋯), ConArg2 (-★-), % Unsuc. instances dynPARTIX (▨).

4 Tests and Discussion

Performance results have been collected on an Intel(R) Core(TM) i7 CPU 970 @3.20GHz (6 core, 2 threads per core), and 16GB of RAM. For both the tools, the output has been redirected to */dev/null*, and the standard error to file. To test dynPARTIX we adopted its 64-bit version, by calling commands like `./dynpartix -f barabasi_graph -s admissible --skept d`. Flag `-f` specifies the input file, `-s` the considered semantics (admissible in this case), and `--skept` sets argument with id d to be checked for sceptical acceptance. We could flag `-n semi`, i.e., the semi-normalised tree-decomposition (more performant than the default normalised one, as stated by the authors of dynPARTIX) only for the admissible semantics, because it is not currently implemented for the other two semantics. We set a timeout of 300 seconds to interrupt the search of each of the two tools.

In Fig. 2 and Fig. 3 we show the tests collected on the whole database presented in Tab. 2, for the credulous and sceptical acceptance respectively. For each of the reported number of nodes (on the x axis), the left y axis reports the CPU time averaged over 100 different instances of AAFs, and 10% random arguments chosen on that class. Acceptance and non-acceptance have been forced to be equally distributed within such random sample (5% each). On the right y axis we also report the percentage of instances that are not solved within the timeout ("% Unsuccessful").

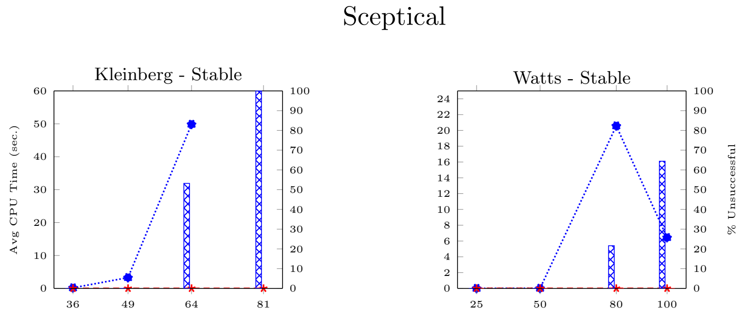


Fig. 3: Avg. time dynPARTIX (⋯●⋯), ConArg2 (-∗-), % Unsuc. instances dynPARTIX (▨).

As we can appreciate from Fig. 2 and Fig. 3, we can state that constraint propagation in ConArg2 works very well with credulous/sceptical acceptance of arguments: most of the problems are solved almost instantaneously, while dynPARTIX is not able to return an answer within the timeout. Note that the performance of dynPARTIX on sceptical acceptance are proven to be better (with low tree-width graphs) or comparable (with high tree-width graphs) to the performance of ASPARTIX, while ASPARTIX works definitely better with high tree-width and credulous acceptance [9].

5 Conclusion

In the paper we have compared two reasoners (dynPARTIX and ConArg2). The main goal has been to study how efficiently state-of-the-art reasoners behave on hard problems related to credulous and sceptical acceptance in lower-order semantics, i.e., admissible, complete, and stable.

In the future we will implement in ConArg2 all the other hard problems related to higher-order semantics [16, Ch. 5]; in particular, credulous/sceptical acceptance in preferred (NP-c/Π_2^P-c), semi-stable (Σ_2^P-c/Π_2^P-c), and stage semantics (Σ_2^P-c/Π_2^P-c), with the purpose to compare our tool with dynPARTIX again, but also with CEGARTIX[8] [11], since it computes sceptical acceptance for the preferred semantics, and both sceptical and credulous acceptance for semi-stable and stage semantics. Moreover, in order to have an engine as more comprehensive as possible, we plan to solve other hard problems (not currently implemented in any other solver) related to the preferred semantics, e.g., its verification (co-NP-complete), and non-emptiness (NP-complete).

To solve higher-order semantics, we will need to work on branch-and-bound search itself, with the result to better manage maximality/minimality of set inclusion directly at the search level; the reason is that it is usually not possible to express such requirements as constraints.

[8] `http://www.dbai.tuwien.ac.at/proj/argumentation/cegartix/`

References

1. A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
2. S. Bistarelli, F. Rossi, and F. Santini. Benchmarking hard problems in random abstract AFs: The stable semantics. In *Computational Models of Argument - Proceedings of COMMA*, volume 266 of *FAIA*, pages 153–160. IOS Press, 2014.
3. S. Bistarelli, F. Rossi, and F. Santini. Efficient solution for credulous/sceptical acceptance in lower-order Dung's semantics. In *26th International Conference on Tools with Artificial Intelligence*, ICTAI, pages 800–804. IEEE Computer Society, 2014.
4. S. Bistarelli, F. Rossi, and F. Santini. Enumerating extensions on random abstract-afs with argtools, aspartix, conarg2, and dung-o-matic. In *Computational Logic in Multi-Agent Systems - 15th International Workshop, CLIMA XV*, volume 8624 of *LNCS*, pages 70–86. Springer, 2014.
5. S. Bistarelli, F. Rossi, and F. Santini. A first comparison of abstract argumentation reasoning-tools. In *ECAI 2014 - 21st European Conference on Artificial Intelligence*, volume 263 of *FAIA*, pages 969–970. IOS Press, 2014.
6. S. Bistarelli and F. Santini. A common computational framework for semiring-based argumentation systems. In *ECAI 2010 - 19th European Conference on Artificial Intelligence*, volume 215 of *FAIA*, pages 131–136. IOS Press, 2010.
7. S. Bistarelli and F. Santini. Conarg: A constraint-based computational framework for argumentation systems. In *23rd International Conference on Tools with Artificial Intelligence*, ICTAI, pages 605–612. IEEE Computer Society, 2011.
8. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–357, 1995.
9. W. Dvořák, M. Morak, C. Nopp, and S. Woltran. dynpartix- A dynamic programming reasoner for abstract argumentation. In *Applications of Declarative Programming and Knowledge Management*, pages 259–268. Springer, 2013.
10. W. Dvorák, R. Pichler, and S. Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artif. Intell.*, 186:1–37, 2012.
11. W. Dvořák, M. Järvisalo, J. P. Wallner, and S. Woltran. Complexity-sensitive decision procedures for abstract argumentation. *Artif. Intell.*, 206:53–78, Jan. 2014.
12. U. Egly, S. A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.
13. P. Erdős and A. Rényi. On the evolution of random graphs. *Bulletin of the International Statistical Institute*, 38(4):343–347, 1961.
14. S. Gabbriellini and P. Torroni. Arguments in social networks. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1119–1120. IFAAMAS, 2013.
15. C. Martel and V. Nguyen. Analyzing Kleinberg's (and other) small-world models. In *Proceedings of the ACM symposium on Principles of distributed computing*, PODC, pages 179–188. ACM, 2004.
16. I. Rahwan and G. R. Simari. *Argumentation in Artificial Intelligence.* Springer Publishing Company, Incorporated, 1st edition, 2009.
17. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence).* Elsevier Science Inc., 2006.
18. D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

Now or Never: Negotiating Efficiently with Unknown Counterparts

Toni Mancini

Computer Science Department, Sapienza University of Rome, Italy
http://tmancini.di.uniroma1.it

Abstract. We define a new protocol rule, Now or Never (NoN), for bilateral negotiation processes which allows self-motivated competitive agents to *efficiently* carry out multi-variable negotiations with remote untrusted parties, where privacy is a major concern and agents know *nothing* about their opponent. By building on the geometric concepts of convexity and convex hull, NoN ensures a continuous progress of the negotiation, thus neutralising malicious or inefficient opponents. In particular, NoN allows an agent to derive in a finite number of steps, and *independently* of the behaviour of the opponent, that there is *no hope* to find an agreement. To be able to make such an inference, the interested agent may *rely on herself only*, still keeping the *highest freedom* in the choice of her strategy.

We also propose an actual NoN-compliant strategy for an automated agent and evaluate the computational feasibility of the overall approach on instances of practical size.

1 Introduction

Automated negotiation among rational agents is crucial in Distributed Artificial Intelligence domains as, e.g., resource allocation [3], scheduling [16], e-business [7], and applications where: (i) no agent can achieve her own goals without interaction with the others (or she is expected to achieve more utility with interaction), and (ii) constraints of various kinds (e.g., security or privacy) forbid the parties to communicate their desiderata to others (the opponent or a trusted authority), hence centralised approaches cannot be used.

We present a framework which allows two *self-motivated*, *competitive* agents to negotiate *efficiently* and find a mutually satisfactory agreement in a particularly *hostile* environment, where each party has *no information* on constraints, preferences, and *willingness to collaborate* of the opponent. This means that also the *bounds* of the domains of the negotiation variables are *not* common knowledge. Our framework deals with negotiations over multiple *constrained* variables over the type of *real numbers*, regarding integer or categorical variables as special cases.

The present setting is very different from what is often assumed in the literature: the set of possible agreements is *infinite* and agents do not even know (or probabilistically estimate) possible opponent's types, variable domain bounds or most preferred values. It is not a split-the-pie game as, e.g., in [6] although with incomplete information, as in [11], and computing *equilibrium* or evaluating *Pareto-optimality* is not possible.

A major problem in our setting is that even *termination* of the negotiation process is not granted: it is in general *impossible* for the single agent to recognise whether the negotiation is making some progress, or if the opponent is just wasting time or arbitrarily delaying the negotiation outcome.

We solve this problem by proposing a new protocol rule, Now or Never (NoN) (Section 3), explicitly designed as to ensure a continuous progress of the negotiation. The rule (whose fulfilment can be assessed *independently* by each party using only the exchanged information) forces the agents to never reconsider already taken decisions, thus injecting a minimum, but sufficient amount of *efficiency* in the process. This leads to the *monotonic shrinking* of the set of possible agreements, which in turn allows each agent to derive in a finite number of steps, *independently* of the behaviour of the opponent, that there is *no hope* to find an agreement.

Furthermore, we discuss the notion of *non-obstructionist* agents, i.e., agents who genuinely aim at efficiently finding an agreement, even sacrificing their preferences (among the agreements they would accept). If both agents are non-obstructionist, the NoN rule guarantees that, whenever the termination condition arises, then *no agreement actually exists*. Hence, in presence of non-obstructionist agents, our approach is both *complete* and *terminates*.

We also propose (Section 4) a full NoN-compliant strategy for an agent which ensures termination independently of the behaviour of the opponent. The strategy, which takes into full account the presence of a utility function on the set of acceptable deals, is inspired to the well-known mechanism of Monotonic Concessions (MC) [13] and allows the agent to perform a sophisticated reasoning, based on the evidence collected so far on the behaviour of the opponent, to select the best deals to offer at each step and keep the process as efficient as possible.

Section 5 specialises NoN to discrete and categorical variables and Section 6 presents experimental results showing that enforcing the NoN rule in practical negotiation instances is computationally feasible.

2 Preliminaries and Negotiation Framework

In the following, we denote with $\mathbb{R}$ the set of real numbers and with $\mathbb{N}^+$ the set of strictly positive integers.

Our framework deals with (possibly multi-deal) negotiations between *two* agents (agent 0 and agent 1) over *multiple constrained* variables. Agents do *not* have any information about constraints, goals, preferences, reasoning capabilities, willingness to collaborate, and strategy of the opponent. *The only knowledge common to both agents is the set of negotiation variables and the protocol rules.*

Definition 1 introduces the main concepts of our framework. Some of them are standard in the literature and are adapted to our framework to ease presentation of the following definitions and results.

Definition 1 (Negotiation process). *A* negotiation process *is a tuple* $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ *where* $\mathcal{V}$ *is a finite set of negotiation* variables*,* $s \in \{0, 1\}$ *is the agent* starting *the negotiation, and* $\mathcal{R}$ *is the set of* protocol rules*.*

The negotiation space *is the multi-dimensional real vector space* $\mathbb{R}^{|\mathcal{V}|}$*. Each point* $D \in \mathbb{R}^{|\mathcal{V}|}$ *is a* deal*. A* proposal *for* π *is a set of at most* k *deals or the* *distinguished element $\perp$. Value $k \in \mathbb{N}^+$ is the* maximum number of deals *that can be included in a single proposal.*

Negotiation proceeds in steps *(starting from step 1) with agents (starting from agent s) alternately exchanging proposals. The proposal exchanged at any step $t \geq 1$ is sent by agent* $\mathrm{ag}(t)$*, defined as s if t is odd and $1 - s$ if t is even.*

The status *of negotiation process π at step $t \geq 1$ is the sequence $\boldsymbol{\mathcal{P}} = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_t$ of proposals exchanged up to step t.*

At each step, the status of π must satisfy the set $\mathcal{R}$ of procotol rules*, a set of boolean conditions on sequences of proposals.*

A strategy *for agent $A \in \{0, 1\}$ for π is a function σ_A that, for each step t such that $\mathrm{ag}(t) = A$ and each status $\boldsymbol{\mathcal{P}} = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_{t-1}$ of π at step $t - 1$, returns the proposal $\mathcal{P}_t$ to be sent by agent A at step t, given the sequence of proposals already exchanged (σ_A is constant for $t = 1$ and $A = s$).*

Our *alternating offers* [14] based framework primarily focuses on *real variables.* In Section 5 we discuss how more specialised domains (e.g., integers, categories) can be handled as special cases, and which is the added value of primarily dealing with real variables. Also, as each proposal can contain up to k deals, our framework supports *multi-deal* negotiations when $k > 1$. Section 4 discusses the added value given by the possibility of exchanging multi-deal proposals.

Protocol rules are important to prevent malicious or inefficient behaviour. Well-designed rules are of paramount importance when the process involves self-motivated and/or unknown/untrusted opponents. For protocol rules to be effective, agents must be able at any time to verify them using the current negotiation status only.

We will use Example 1 as a running example throughout the paper.

Example 1 (Alice vs. Bob). Alice wants to negotiate with her supervisor *Bob* to schedule a meeting. At the beginning, agents agree on the relevant variables $\mathcal{V}$: (i) the start day/time t; (ii) the meeting duration d.

Deals are assignments of values to variables, as, e.g., $D = \langle t =$ "Mon 11 am", $d =$ "30 min"$\rangle$. Deals can be easily encoded as points in $\mathbb{R}^2$.

Definition 2 (Negotiation outcomes). *Let $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ be a negotiation process whose status at step $T > 1$ is $\boldsymbol{\mathcal{P}} = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_T$. We say that π* terminates *at step T if and only if T is the smallest value such that one of the following two cases holds:*

- success*: $\mathcal{P}_T = \{D\} \subseteq \mathcal{P}_{T-1}$ (*$\mathrm{ag}(T)$ *accepts deal D proposed by* $\mathrm{ag}(T-1)$ *at step $T - 1$)*
- opt-out*: $\mathcal{P}_T = \perp$ (*$\mathrm{ag}(T)$ *opts-out).*

If no such a T exists, then π is non-terminating *(*non-term*).*

Success*,* opt-out *and* non-term *are the possible negotiation* outcomes.

A negotiation process can be infinite (case *non-term*) or terminate in a finite number of steps, either with an *agreement* found (case *success*, where point D is the *agreement*) or with a failure (case *opt-out*, where one of the agents proposes $\perp$, which aborts the process).

For a deal to be *acceptable* to an agent, some *constraints* must be satisfied. Such constraints, which are *private information* of the single agent, are formalised by Definition 3.

Definition 3 (Feasibility region). *Let $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ be a negotiation process. The* feasibility region *of agent $A \in \{0, 1\}$, denoted by R_A, is the subset of the negotiation space $\mathbb{R}^{|\mathcal{V}|}$ of deals* acceptable *to A.*

For agent A, any deal in R_A is better than failure. An *agreement* is thus any deal $D \in R_0 \cap R_1$.

Example 2 (Alice vs. Bob (cont.)). Alice wants the meeting no later than Wednesday. Normally she needs at least 30 minutes and does not want the meeting to last more than one hour; however, if she has to wait until Wednesday, she would have time during her Tuesday's trip to prepare new material to show; in this case she wants the meeting to last at least one hour, but no more than 75 minutes. Conversely, Bob has his own, private, constraints.

Fig. 1a shows Alice's feasibility region in a 2D space, as the areas delimited by the three polygons. The region takes into account duties in her agenda (e.g., Alice is busy on Monday from 1pm to 4pm).

Agents may have *preferences* on the deals in their feasibility region. Such preferences are often represented by a private *utility function.* Fig. 1a shows that, e.g., Alice prefers a long meeting on Monday. We will handle the agent utility function in Section 4 when we present a full strategy for an agent.

We assume (as typically done, see, e.g., [5,12]) that agents offer only deals in their feasibility region (i.e., agents do not offer deals they are not willing to accept). This does not limit our approach, as suitable ex-post measures (e.g., penalties) can be set up to cope with the case where a deal offered by an agent (but *not* acceptable to her) is accepted by the other.

3 Now or Never

We are interested in negotiations which are *guaranteed* to terminate in a finite number of steps (note that, being negotiation variables real-valued, the set of potential agreements is infinite), so we want to avoid case *non-term* of Definition 2. In this section we define a protocol rule, the Now or Never (NoN) rule, which is our key to drive a negotiation process towards termination, avoiding malicious or inefficient agents behaviour. The rule relies on the notions of Definition 4.

Definition 4 (Convex region, convex hull, operator $\cup\kern-0.55em{\scriptstyle\diamond}$). *Let $\mathbb{R}^n$ be the n-dimensional real vector space (for any $n > 0$). Region $R \subseteq \mathbb{R}^n$ is* convex *if, for any two points D_1 and D_2 in R, the straight segment $\overline{D_1 D_2}$ is entirely in R.*

Given a finite set of points $\mathcal{D} \subset \mathbb{R}^n$, the convex hull *of $\mathcal{D}$,* conv$(\mathcal{D})$*, is the smallest convex region of $\mathbb{R}^n$ containing $\mathcal{D}$.*

Given a collection of finite sets of points $\boldsymbol{\mathcal{D}}$, $\cup\kern-0.55em{\scriptstyle\diamond}\,\boldsymbol{\mathcal{D}}$ is the union of the convex hulls *of all sets in $\boldsymbol{\mathcal{D}}$: $\cup\kern-0.55em{\scriptstyle\diamond}\,\boldsymbol{\mathcal{D}} = \bigcup_{\mathcal{D} \in \boldsymbol{\mathcal{D}}} \{\text{conv}(\mathcal{D})\}$.*

Convexity arises often in feasibility regions of agents involved in negotiations. An agent feasibility region is convex if, for any two acceptable deals D_1 and D_2, all *intermediate* deals (i.e., those lying on $\overline{D_1 D_2}$) are acceptable as well. In some cases [2,12,4] the feasibility region of an agent is *entirely* convex (consider, e.g., a negotiation instance over a single variable, the price of a good). In other cases this does not hold. However, a feasibility region may always be considered as the *union* of a number of convex sub-regions. Furthermore, in most real cases, this number is *finite* and *small.* Also, in most practical situations, the closer two acceptable deals D_1 and D_2, the higher the likelihood that intermediate deals are acceptable as well.

Example 3 (Alice vs. Bob (cont.)). Knowing that deals $\langle t =$ *"Mon at 11am"*, $d =$ *"30 min"*$\rangle$ and $\langle t =$ *"Wed at 3pm"*, $d =$ *"1 hour"*$\rangle$ are both acceptable to Bob would not be a strong support for Alice to assume that also $\langle t =$ *"Tue at 1pm"*, $d =$ "45 min"$\rangle$ would be acceptable to him. On the other hand, if $\langle t =$ *"Mon at 9am"*, $d =$ *"40 min"*$\rangle$ and $\langle t =$ *"Mon at 9.30am"*, $d =$ *"20 min"*$\rangle$ are both acceptable to Bob, it would not be surprising if also $\langle t =$ *"Mon at 9.15am"*, $d =$ *"30 min"*$\rangle$ is acceptable.

Before formalising the NoN rule (Definition 6), we introduce it using our example.

Example 4 (Alice vs. Bob (cont.)). Steps below are shown in Fig. 1.

Steps 1 and 2. Alice starts the negotiation by sending proposal $\mathcal{P}_1 = \{A_1^a, A_1^b\}$. As a reply, she receives $\mathcal{P}_2 = \{B_2^a, B_2^b\}$ (see Fig. 1b). As none of Bob's counteroffers, B_2^a and B_2^b, belong to $\mathrm{conv}(\{A_1^a, A_1^b\}) = \overline{A_1^a A_1^b}$, all such deals are *removed* from further consideration (by exploiting NoN). The rationale is as follows:

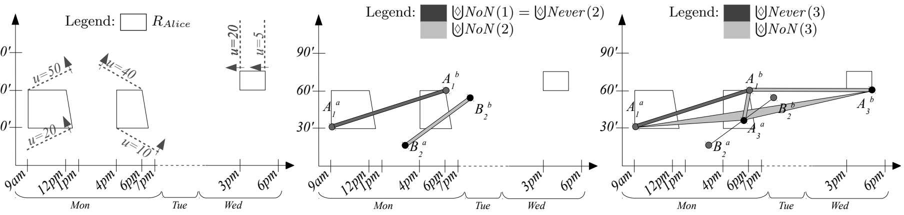
(a) Bob had *no evidence* that $\mathrm{conv}(\{A_1^a, A_1^b\})$ includes deals outside R_{Alice} (i.e., at the end of step 1 Bob had no evidence that this portion of R_{Alice} is not convex).

(b) Given that Bob has not proposed any such deal therein, then either $R_{Bob} \cap \mathrm{conv}(\{A_1^a, A_1^b\}) = \emptyset$ (in which case, Bob has no interest at all in proposing there), or Bob has chosen not to go for any such a deal *now* (as, e.g., he currently aims at higher utility).

(c) In the latter case, NoN forbids Bob to reconsider that decision anymore (*never*).

Step 3. Alice, having *no evidence* that $\mathrm{conv}(\{B_2^a, B_2^b\})$ includes deals outside R_{Bob}, proposes $\mathcal{P}_3$ containing deal $A_3^a \in \mathrm{conv}(\{B_2^a, B_2^b\}) \cap R_{Alice}$ (see Fig. 1c): by proposing A_3^a she aims at closing the negotiation successfully *now*, believing that such a deal (intermediate to B_2^a and B_2^b) is likely to be acceptable also to Bob. Alice also includes in $\mathcal{P}_3$ deal A_3^b.

Step 4. It's Bob's turn again. By receiving $\mathcal{P}_3 = \{A_3^a, A_3^b\}$, Bob knows that such deals belong to R_{Alice}. Assume that Bob rejects $\mathcal{P}_3$ by sending a counteroffer. As there is no evidence that $\mathrm{conv}(\{A_1^a, A_3^a, A_3^b\})$, $\mathrm{conv}(\{A_1^b, A_3^b\})$, or $\mathrm{conv}(\{A_1^b, A_3^a\})$ (the 3 light-grey areas in Fig. 1c) include deals outside R_{Alice}, NoN forces him to take a decision: either his counteroffer $\mathcal{P}_4$ contains some deals in one of such regions, or he must forget those regions forever. Note that NoN does not apply to, e.g., $\mathrm{conv}(\{A_1^b, A_3^a, A_3^b\})$, as this region contains B_2^b, which was part of a Bob's proposal already rejected by Alice. Hence, there is already evidence that some of the deals in $\mathrm{conv}(\{A_1^b, A_3^a, A_3^b\})$ are acceptable to Bob and NoN does not forbid agents to further explore that region.



(a) Alice's region and utility (b) End of step 2 (c) End of step 3

Fig. 1: Alice vs. Bob (Example 4)

Definition 5 (Sets *Never* and *NoN*). *Let $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ be a negotiation process and $\boldsymbol{\mathcal{P}} = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_T$ its status at step $T \geq 1$. For each agent $A \in \{0,1\}$ and each step $1 \leq t \leq T$, let $\mathrm{deals}_A(t)$ be the set of all the deals in $\boldsymbol{\mathcal{P}}$ proposed by A up to step t (included). Sets $Never(t)$ and $NoN(t)$ are defined inductively for each $t \geq 1$ as follows:*

t=1: *$Never(1) = \emptyset$, $NoN(1) = \{\mathcal{P}_1\}$*

t>1:

$$Never(t) = \begin{cases} Never(t{-}2) \cup NoN(t{-}1) & \text{if } \mathcal{P}_t \cap \biguplus NoN(t{-}1) = \emptyset \\ Never(t{-}2) \cup \{\{D\} \mid D \in NoN(t{-}1)\} & \text{otherwise} \end{cases}$$

$$NoN(t) = \left\{\mathcal{D} \subseteq \mathrm{deals}_{\mathrm{ag}(t)}(t) \middle| \operatorname{conv}(\mathcal{D}) \cap \biguplus Never(t) = \emptyset\right\}$$

where $\mathcal{P}_t$ is the proposal sent by $\mathrm{ag}(t)$ at step t and $Never(0) = \emptyset$.

At each step t, $\biguplus NoN(t)$ represents the region, defined by $\mathrm{ag}(t)$'s deals, for which the other agent $1 - \mathrm{ag}(t)$ needs, in the next step $(t+1)$ to take a NoN decision: to offer a deal therein (showing to $\mathrm{ag}(t)$ that she is potentially interested to that region) or to neglect that region forever. Similarly, $\biguplus Never(t)$ represents the region, defined by $(1 - \mathrm{ag}(t))$'s deals, for which $\mathrm{ag}(t)$ has taken a *never* decision. Deals therein cannot be offered any more. Note that $\biguplus Never(t) \supseteq \biguplus Never(t-2)$ for all $t \geq 2$ (i.e., sequences $Never(t)$ for odd and even values of t are monotonically non-decreasing). Fig. 1 shows *NoN* and *Never* regions at all steps of the previous example.

Definition 6 formalises our NoN protocol rule, which forbids agents to reconsider *never* decisions already taken.

Definition 6 (Now or Never rule). *Status $\boldsymbol{\mathcal{P}} = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_T$ of negotiation process $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ satisfies the NoN protocol rule if, for all steps $2 \leq t \leq T$, $\mathcal{P}_t \cap \biguplus Never(t-2) = \emptyset$.*

Proposition 1 shows that the NoN rule of Definition 6 allows agents to infer when no further agreements are possible. All proofs are omitted for lack of space.

Proposition 1 (Termination condition). *Let $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ be a negotiation process where the NoN rule is enforced and let $\boldsymbol{\mathcal{P}} = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_T$ be the status of π at step $T \geq 2$.*

If $R_{\mathrm{ag}(T)} \subseteq \biguplus Never(T-1) \cup \biguplus Never(T-2)$ and $\mathcal{P}_T$ is not a singleton $\{D\} \subseteq \mathcal{P}_{T-1}$, then:

(a) there exists no extension $\boldsymbol{\mathcal{P}}' = \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{T-1}, \mathcal{P}_T, \ldots, \mathcal{P}_{T'}$ of $\boldsymbol{\mathcal{P}}$ to step $T' > T$ such that $\mathcal{P}_{T'} = \{D\} \subseteq \mathcal{P}_{T'-1}$

(b) for all $D \in R_0 \cap R_1$, there exists $1 < t_D < T$ such that $D \in \bigcup\!\!\!\!\!\diamond\, NoN(t_D - 1) \cap \bigcup\!\!\!\!\!\diamond\, Never(t_D)$.

A consequence of (a) is that, if at step $T \geq 2$, $R_{\mathrm{ag}(T)} \subseteq \bigcup\!\!\!\!\!\diamond\, Never(T-1) \cup \bigcup\!\!\!\!\!\diamond\, Never(T-2)$ and *agent* $\mathrm{ag}(T)$ *cannot or does not want to accept a deal offered in the last incoming proposal $\mathcal{P}_{T-1}$, she can safely opt-out* by proposing $\mathcal{P}_T = \perp$, as she has *no hope* to reach an agreement in the future. Also, from (b), for every mutually acceptable agreement D, there was a step $t_D < T$ in which agent $\mathrm{ag}(t_D)$ took a *never* decision on a NoN region containing D. This means that $\mathrm{ag}(t_D)$, although knowing that $D \in R_{\mathrm{ag}(t_D)}$ was likely to be acceptable also to the opponent (because she had no evidence, at that time, that the portions of the opponent region defined by deals in $NoN(t_D - 1)$ were not convex), *explicitly decided* not to take that chance and proposed elsewhere.

As a matter of fact, NoN can be thought as a *deterrent*, for each agent, to delay the negotiation by ignoring plausible agreements which, although acceptable to her, do not grant herself the utility she currently aims at. As NoN forbids the agents to propose such deals in the future, any such "obstructionist" behaviour has a price in terms of opportunities that must be sacrificed forever.

Definition 7 defines *non-obstructionist agents.*

Definition 7 (Non-obstructionist agent). *Let $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ be a negotiation process where the NoN rule is enforced. Agent $A \in \{0, 1\}$ is* non-obstructionist *if her strategy satisfies the following conditions for all $t \geq 2$ such that $\mathrm{ag}(t) = A$:*

1. *if $\mathcal{P}_{t-1} \cap R_A \neq \emptyset$, then $\mathcal{P}_t = \{D\} \subseteq \mathcal{P}_{t-1}$*
2. *else if $\bigcup\!\!\!\!\!\diamond\, NoN(t-1) \cap R_A \neq \emptyset$, then $\mathcal{P}_t \cap \bigcup\!\!\!\!\!\diamond\, NoN(t-1) \neq \emptyset$.*

A non-obstructionist agent A accepts *any* acceptable deal $D \in R_A$ and takes a *now* decision at all steps t when $\bigcup\!\!\!\!\!\diamond\, NoN(t-1)$ intersects R_A. Non-obstructionist agents genuinely aim at finding an agreement *efficiently*, even sacrificing their preferences among deals they would accept. However, they are *not* necessarily collaborative, as they do not disclose to the opponent their constraints and preferences.

Proposition 2 shows that, in a negotiation process between two non-obstructionist agents, if one of the parties reaches the *termination condition* of Proposition 1, then no agreement exists (i.e., $R_0 \cap R_1 = \emptyset$).

Proposition 2 (Completeness). *Let $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ be a negotiation process between two non-obstructionist agents where the NoN rule is enforced.*

If π reaches, at step $T - 1 \geq 2$, status $\boldsymbol{\mathcal{P}} = \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_{T-1}$ s.t. $R_{\mathrm{ag}(T)} \subseteq \bigcup\!\!\!\!\!\diamond\, Never(T-1) \cup \bigcup\!\!\!\!\!\diamond\, Never(T-2)$, then $R_0 \cap R_1 = \emptyset$.

4 A Terminating Strategy Based on Monotonic Concessions

Propositions 1 and 2 show that the Now or Never (NoN) rule allows each agent to detect when the negotiation process can be safely terminated, as no agreement can be found in the sequel. However, still the termination condition may not arise in a finite number of steps. In this section we show that, with NoN, *termination can be enforced by any agent alone, without relying on the willingness to terminate of the counterpart.* To this end, from now on we focus on one agent only, which we call agent A (A can be either 0 or 1). To ease presentation, the other agent, agent $1 - A$, will be called agent B.

We make some assumptions on the feasibility region of agent A: (a) R_A is *bounded* and defined as the union $P_1 \cup \cdots \cup P_q$ of a *finite* number q of convex sub-regions; (b) each convex sub-region P_i ($1 \leq i \leq q$) of R_A is defined by *linear constraints*, hence is a (bounded) *polyhedron* in $\mathbb{R}^{|\mathcal{V}|}$. *Any* bounded feasibility region can be approximated arbitrarily well with a (sufficiently large) union of bounded polyhedra. However, in many practical cases, a finite and small number of polyhedra suffices.

Deals in R_A may not be equally worth for agent A, who may have a (again, *private*) utility function u_A to *maximize*. We assume that u_A is *piecewise-linear* and defined (without loss of generality) by a linear function u^i_A for each polyhedron P_i of R_A ($1 \leq i \leq q$). For this definition to be well founded, if a deal D belongs to two different polyhedra P_i and P_j of R_A, it must be $u^i_A(D) = u^j_A(D)$. Note that, again, any differentiable utility function can be approximated arbitrarily well with a piecewise-linear utility, provided R_A is decomposed in an enough number of polyhedra.

In this setting, we define a full strategy for agent A for negotiation processes $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ for which $k \geq 2$, i.e., in which exchanged proposals can contain multiple deals. Although our strategy is correct independently of the opponent region shape, it is designed for the common cases where agent A believes that the opponent feasibility region is the union of a small number of convex sub-regions (not necessarily polyhedra). Hence, a task of agent A while following the strategy is to *discover* non-convexities of the opponent region during negotiation and take them into account.

Our strategy is inspired by (but different from) the well-known mechanism of Monotonic Concessions (MC) [13]. It has three phases, *utility-driven*, *non-obstructionist*, and *terminating* phases, which are executed in the given order.

4.1 Utility-Driven Phase

Agent A keeps and dynamically revises two utility thresholds, α and u, which are, respectively, the *responding* and the *proposing threshold*. At each step t such that $\mathrm{ag}(t) = A$, agent A uses: (a) threshold α to decide whether to take a *now* decision (if $t > 1$), by including, in the proposal $\mathcal{P}_t$ she will propose next, a deal in $\bigcup\mkern-14mu\diamond\, \mathit{NoN}(t-1)$ (possibly accepting one deal in $\mathcal{P}_{t-1}$), and (b) threshold u to select the other deals to include in $\mathcal{P}_t$ ($t \geq 1$).

By generalising [6,12], α is a function of the agent A utility of the best deal D_{next} that would be chosen in step (b). In particular, α is $u_A(D_{\text{next}}) - \mathit{span} \cdot \xi$, where *span* is the absolute difference of the extreme values of u_A in R_A and $0 \leq \xi \leq 1$ is a parameter (possibly varying during the negotiation) called *respond policy*. Hence, if $\xi = 1$, agent A accepts all acceptable deals and takes a *now* decision whenever possible, behaving in a non-obstructionist way (Definition 7). On the other extreme, if $\xi = 0$ the agent accepts only incoming deals $D \in R_A$ that are not worse than the best proposal D_{next} that would be chosen next in step (b), upon rejection of D.

Our strategy for this phase is decomposed into *responding*, *proposing* and *conceding* sub-strategies as in [8], after an *initialisation* phase where agent A sets u to the highest utility of deals in R_A (as in the spirit of MC).

Responding. At step $t \geq 2$, after that agent A has received proposal $\mathcal{P}_{t-1} \neq \perp$, proposal $\mathcal{P}_t$ is chosen as follows. Let $R_A^\alpha = \{D \in R_A \mid u_A(D) \geq \alpha\}$.

(1) If $\mathcal{P}_{t-1}$ contains deals in $R_A^\alpha - \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-2)$, then $\mathcal{P}_t = \{D\}$, where D is one such a deal giving agent A the highest utility (i.e., agent A accepts the best deal D among those acceptable in $\mathcal{P}_{t-1}$ granting herself at least utility α). Otherwise:

(2) $\mathcal{P}_t$ contains a deal in $(\mathop{\bigcup\kern-0.75em\diamond} \mathit{NoN}(t-1) \cap R_A^\alpha) - \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-2)$ if and only if this region is not empty (*now* decision taken).

Given that the closer deals in a set $\mathcal{D}$ defining $\mathit{NoN}(t-1)$ (see Definition 5) the more likely they belong to a single convex sub-region of R_B, as for (2) agent A selects a deal with the highest utility in a set $\mathcal{D}$ having the minimum *diameter*.

Proposing. At any step $t \geq 1$ such that $\mathrm{ag}(t) = A$, if agent A has not accepted an incoming deal (case (1) of the *responding* sub-strategy), proposal $\mathcal{P}_t$ contains *additional* deals (as to make $|\mathcal{P}_t| = k \geq 2$). Let $R_A^u = \{D \in R_A \mid u_A(D) \geq u\}$ (which is again a union of bounded polyhedra, as u_A is piecewise-linear). Deals to be proposed in $\mathcal{P}_t$ are carefully selected among *vertices* of R_A^u (some of them can be vertices of the overall region R_A) which do not belong to $\mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-2)$, as agent A needs to comply with the NoN rule. If $t > 1$, vertices of R_A^u to be proposed will be carefully selected by reasoning on the *evidence* provided by the past opponent behaviour. The reasoning is as follows.

Let $\hat{n}(t)$ be the *minimum* number of convex sub-regions that *must* compose $R_B - \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-1)$, i.e., the opponent region minus the regions for which the opponent has taken a *never* decision (and in which, by the NoN rule, no agreements can be found in the sequel): $\hat{n}(t)$ is the minimum value such that there exists a $\hat{n}(t)$-partition $\{\mathcal{D}_1, \ldots, \mathcal{D}_{\hat{n}(t)}\}$ of $\mathrm{deals}_B(t-1)$ (i.e., a mapping of each opponent deal to one sub-region) such that for all $1 \leq j \leq \hat{n}(t)$, $\mathrm{conv}(\mathcal{D}_j) \cap \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-1) = \emptyset$.

Agent A temporarily focuses on $\hat{n}(t)$, assuming that $R_B - \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-1)$ is the union of *exactly* $\hat{n}(t)$ convex sub-regions. We call this assumption *Non-obstructionist Opponent Assumption (NOA)*. Under NOA, agent A tries to regard the past opponent behaviour as non-obstructionist, hence interprets the already taken *never* decisions as an admission that $R_B \cap \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-1) = \emptyset$ (Proposition 2). Value $\hat{n}(t)$ is the minimum number of convex sub-regions that must compose R_B which is consistent with this (optimistic) hypothesis.

Agent A computes the subsets $\mathcal{D}$ of the opponent deals that *might* belong to the same convex sub-region of $R_B - \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-1)$, provided that NOA is correct. We call these sets of deals *Possible Opponent Clusters (POCs)*:

$$\boldsymbol{\mathcal{K}}(t) = \left\{ \mathcal{D} \subseteq \mathrm{deals}_B(t-1) \;\middle|\; \begin{array}{l} \exists\ \hat{n}(t)\text{-partition } \{\mathcal{D}_1, \ldots, \mathcal{D}_{\hat{n}(t)}\} \text{ of } \mathrm{deals}_B(t-1) \\ \text{s.t. } \forall j \in [1, \hat{n}(t)]\ \ \mathrm{conv}(\mathcal{D}_j) \cap \mathop{\bigcup\kern-0.75em\diamond} \mathit{Never}(t-1) = \emptyset \end{array} \right\} \qquad (1)$$

Let $\mathrm{proj}(R, R')$ (the *projection* of region R onto R') be the set of points X for which there exists $Y \in R$ such that $\overline{XY}$ intersects R' [2]. Region $\mathrm{proj}(R, R')$ is an unbounded polyhedron if both R and R' are polyhedra (see Fig. 2a, where $\mathrm{proj}(R, R')$ is the unbounded grey area) and $\mathrm{proj}(R, R' \cup R'') = \mathrm{proj}(R, R') \cup$

$\mathrm{proj}(R, R'')$. Provided that NOA is correct, agent A can derive (Proposition 3) that region

$$\mathit{\Pi}(t) = \bigcap\nolimits_{\mathcal{D} \in \boldsymbol{\mathcal{K}}(t)} \mathrm{proj}(\mathrm{conv}(\mathcal{D}), \cup\!\!\!\!\!\diamond\,\mathit{Never}(t-1))$$

does not contain agreements that can be still reached.

Proposition 3. *If, at step $t \geq 3$ s.t. $\mathrm{ag}(t) = A$, NOA is correct, then $\mathit{\Pi}(t) \cap (R_B - \cup\!\!\!\!\!\diamond\,\mathit{Never}(t-1)) = \emptyset$.*

Example 5 (Alice vs. Bob (cont.)). Consider Fig. 2b. At step 4 Bob sent Alice proposal $\mathcal{P}_4 = \{B_4^a\}$. At step 5 (Alice's turn), $\hat{n}(5)$ is 3, as it is clear that B_2^a, B_2^b, and B_4^a belong to all-different convex sub-regions of $R_{Bob} - \cup\!\!\!\!\!\diamond\,\mathit{Never}(4)$. POCs are $\boldsymbol{\mathcal{K}}(5) = \{\{B_2^a\}, \{B_2^b\}, \{B_4^a\}\}$. Region $\mathit{\Pi}(5)$ is the area in light-grey: if NOA is correct ($R_{Bob} - \cup\!\!\!\!\!\diamond\,\mathit{Never}(4)$ or, equivalently, R_{Bob} if Bob is non-obstructionist, consists of exactly 3 convex sub-regions), then no $X \in \mathit{\Pi}(5)$ can belong to $R_{Bob} - \cup\!\!\!\!\!\diamond\,\mathit{Never}(4)$.

Besides always ignoring vertices in $\cup\!\!\!\!\!\diamond\,\mathit{Never}(t-2)$ (as to comply with the NoN rule), as a result of Proposition 3 agent A (exploiting NOA) can *temporarily ignore* vertices of R_A^u in $\mathit{\Pi}(t)$ while choosing deals to propose at step t. By exploiting the fail-first principle, we define the following criterion (*best vertex under NOA*) to select the next vertices in $R_A^u - \mathit{\Pi}(t)$ (and not in $\cup\!\!\!\!\!\diamond\,\mathit{Never}(t-2)$) to propose: those that, if rejected, would make the *highest* number of vertices be excluded in the next step, under NOA.

Conceding. When no more vertices in $R_A^u - \mathit{\Pi}(t)$ (and not in $\cup\!\!\!\!\!\diamond\,\mathit{Never}(t-2)$) can be proposed, agent A reduces threshold u, if possible, by a given amount $\mathit{\Delta}u$, whose value, possibly varying during time (see, e.g, [5]), depends on the application. Reducing u is in the spirit of MC (where the agent *increases* during time the *opponent* utility of the proposed deals). Differently from MC, here agent A *reduces own* utility of the deals she proposes (with the goal of approaching opponent's demand), as she has no information about opponent utility.

Let $\hat{T}$ ($\mathrm{ag}(\hat{T}) = A$) be the step in which agent A reduces u and R_A^u becomes equal to R_A (i.e., u cannot be further reduced). From step $\hat{T}$ onwards, the strategy of agent A moves to the *non-obstructionist* phase.

4.2 Non-Obstructionist Phase

Our strategy for this phase is decomposed into *responding* and *proposing* sub-strategies. As utility threshold u has already reached its minimum, in this phase there is no *conceding* sub-strategy.

Responding. The *responding* sub-strategy is identical to that of the *utility-driven* phase with $\alpha = u$. Given that in the non-obstructionist phase u is at its minimum, agent A accepts any incoming acceptable deal and takes a *now* decision whenever possible. Thus, the agent is now certainly non-obstructionist, independently of the value of her respond policy ξ.

Proposing. As a result of acting in a non-obstructionist way, from step $\hat{T}$ onwards the following result holds:

Proposition 4. *For each step $t \geq \hat{T}$ such that* $\mathrm{ag}(t) = A$*,* $R_A \cap \cup\!\!\!\!\diamond\, \mathit{Never}(t-2) = R_A \cap \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)$.

Hence, for each step $t \geq \hat{T}$ such that $\mathrm{ag}(t) = A$, if agent A has not accepted an incoming deal, the region in which the additional deals to propose will be selected (as to make $|\mathcal{P}_t| = k \geq 2$ whenever possible), i.e., $R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(t-2)$, is steadily equal to $R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)$.

In this phase, agent A aims at proposing *vertices* of $R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)$ with the goal of eventually covering it within the *never* set of the opponent, as to reach the termination condition of Proposition 1. Unfortunately, as both R_A and $\cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)$ are unions of polyhedra, their difference might *not* be represented as a union of polyhedra. Anyway, it can be always represented as a union of Not Necessarily Closed polyhedra (i.e., polyhedra possibly defined by some *strict* inequalities, with some of their faces and vertices *not* belonging to them). In order to comply with the NoN rule, the agent must not propose vertices of $R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)$ not belonging to that region, as they would belong to $\cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)$. The problem is solved by computing a suitable *under-approximation* $\lfloor R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2) \rfloor \subseteq R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)$ which can be defined as a union of bounded (and closed) polyhedra. Note that such an under-approximation can be computed in order to make the error

$$R_A^{\mathrm{err}} = (R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2)) - \lfloor R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2) \rfloor$$

arbitrarily small. As a special case, if agent A was non-obstructionist from the beginning of the negotiation process, $R_A \cap \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2) = \emptyset$ and $R_A^{\mathrm{err}} = \emptyset$.

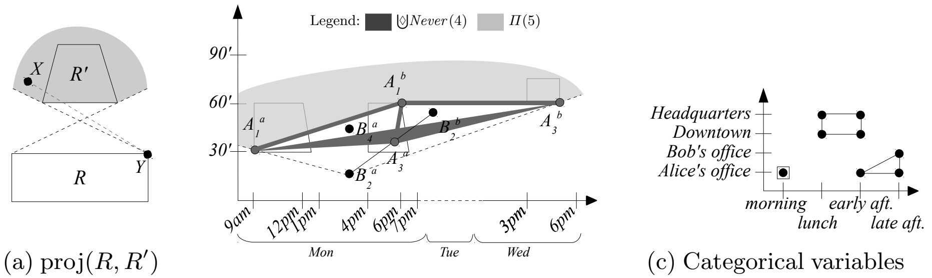
Agent A continues to use both NOA and $\Pi(t)$ as defined in the *utility-driven* phase. In particular, the agent proposes *vertices* of $\lfloor R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2) \rfloor$ which are not in $\Pi(t)$. When no more such vertices can be proposed, NOA is gradually relaxed (i.e., $\hat{n}(t)$ is gradually increased) and the remaining vertices of $\lfloor R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2) \rfloor$ are enabled. By construction, $\hat{n}(t)$ cannot grow beyond the number of deals proposed by the opponent so far. If also in that case $\Pi(t)$ covers $\lfloor R_A - \cup\!\!\!\!\diamond\, \mathit{Never}(\hat{T}-2) \rfloor$, the agent sets $\Pi(t)$ to $\cup\!\!\!\!\diamond\, \mathit{Never}(t-1)$, hence assumes that R_B consists of at least one convex sub-region *not yet* disclosed by the opponent (i.e., not containing any of the past incoming deals).

As it happens in the *utility-driven* phase, given that *multi-deal* proposals are allowed ($k \geq 2$), all vertices will be proposed within a finite number of steps independently of the number of *now* decisions taken. When all vertices have been proposed and no agreement has been reached, agent A enters the *terminating phase.*

4.3 Terminating Phase

In this phase, agent A continues by sending *empty proposals* until she receives and accepts an acceptable deal or infers $R_A - R_A^{\mathrm{err}} \subseteq \cup\!\!\!\!\diamond\, \mathit{Never}(T-1) \cup \cup\!\!\!\!\diamond\, \mathit{Never}(T-2)$. Proposition 5 states that also this condition will arise in a finite number of steps.

Proposition 5. *Let $\pi = \langle \mathcal{V}, s, k, \mathcal{R} \rangle$ be a negotiation process ($k \geq 2$) where the NoN rule is enforced. If any agent $A \in \{0, 1\}$ uses the strategy above, then, within a finite number of steps $T \geq \hat{T} \geq 2$ such that* $\mathrm{ag}(T) = A$*, either an agreement is found or condition $R_A - R_A^{\mathrm{err}} \subseteq \cup\!\!\!\!\diamond\, \mathit{Never}(T-1) \cup \cup\!\!\!\!\diamond\, \mathit{Never}(T-2)$ is satisfied.*



(a) proj(R, R') (grey area)

(b) Alice vs. Bob (end of step 4)

(c) Categorical variables

Fig. 2

Condition of Proposition 5 can be considered the *termination condition* of Proposition 1 in case agent A had admissible region $R_A - R_A^{\mathrm{err}}$. Given that region R_A^{err} can be chosen as to be *arbitrarily small*, agent A can terminate the negotiation when this condition is reached. Any possible remaining acceptable deals would be in the (arbitrarily small) region R_A^{err}.

We stress again that, in case agent A is non-obstructionist from the beginning, for all $t \geq \hat{T}$ such that $\mathrm{ag}(t) = A$, R_A^{err} can be made *empty*. Hence, as it happens for any acceptable deal in $R_A \cap \uplus \mathit{Never}(t-2) = R_A \cap \uplus \mathit{Never}(\hat{T}-2)$, any acceptable deal in R_A^{err} can be considered as an opportunity (with arbitrarily small Euclidean distance to $R_A \cap \uplus \mathit{Never}(\hat{T}-2)$) that agent A had to sacrifice for having behaved in an obstructionist way (at most) up to step $\hat{T}-2$.

5 Handling Discrete and Categorical Variables

The NoN rule works also when (some of) the variables are discrete (e.g., integer), if we consider the union of the *integer hulls* [15] of the polyhedra in the *NoN* and *Never* sets of Definition 5. Integer Linear Programming results tell us that the integer hull of a polyhedron can still be represented with linear (plus integrality) constraints. Vertices of this new polyhedron have integer coordinates. Hence, the NoN rule as well as the strategy above and the underlying projection-based reasoning can be adapted to *prune* the space of the possible agreements: only the branches that deal with *now* decisions need to be refined (deals in $\uplus \mathit{NoN}(t-1)$ proposed at step t need to have integer coordinates). Also, $R_A - \uplus \mathit{Never}(\hat{T}-2)$ can always be represented by an union of closed polyhedra, hence R_A^{err} can be always made empty. Categorical variables can be tackled by fixing an ordering of their domain (common to both parties) and mapping them onto integers. Fig. 2c shows Alice's region in a variation of Example 1 where variables are categorical.

6 Implementation and Experiments

Our framework has at its core well-studied tasks in computational geometry and (integer) linear programming [15]. However, to our knowledge, the exact complexity of the agent's reasoning is unknown, as these core tasks must be repeated on sets of exchanged deals. Still, existing libraries of computational geometry algorithms can manage the size of instances needed for practical scenarios. We have implemented a system that uses the Parma Polyhedra Library (PPL) [1] to compute polyhedra, convex hulls, and projections, and an all-solutions SAT solver [9] to revise $\hat{n}(t)$ and Possible Opponent Clusters (POCs) (the problem is reduced to hypergraph-colouring). Performance on these sub-problems are very good: PPL completes most of the required tasks within very few seconds (on a reasonably small set of variables, e.g., 3–4) and the generated SAT instances are trivial. Although the number of sets in *NoN* and *Never* can grow exponentially, by keeping only (depending on the case) their $\subseteq$-maximal or $\subseteq$-minimal members (which is enough to enforce the NoN rule and to perform the needed reasoning), the overall memory requirements become, in the instances we consider below, compatible with the amount of RAM available on an ordinary PC.

In the following we present an empirical evaluation of the *computational feasibility* of the approach. Negotiations have been performed between two identical agents. We evaluated our implementation on both random and structured instances using a *single* computer (a PC with a dual-core AMD Opteron 3GHz and 8GB RAM) for both agents. At each step, agents can exchange contracts of at most $k = 2$ deals. Note that, as our approach requires agents to comply with the NoN rule, it *cannot* be evaluated against other negotiators.

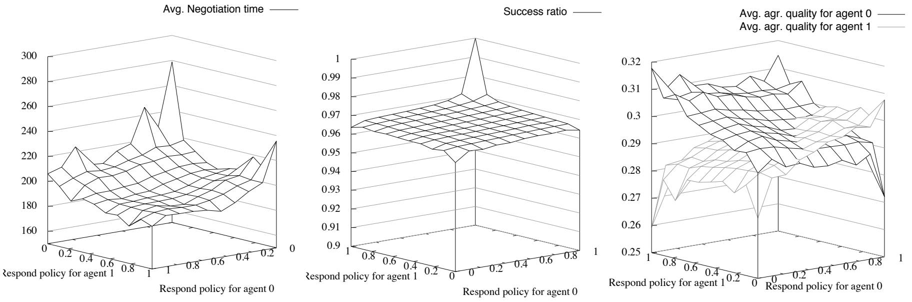
Random Instances. We generated 100 random negotiation instances over 3 variables. Feasibility regions are unions of 3 random polyhedra, each with at most 10 vertices. In about 44% of the instances $R_0 \cap R_1 \neq \emptyset$. The average volume of the intersection is 2.19% of the volume of each agent's region (stddev is 4.5%). Agents have random piecewise-linear utilities and concede constant $\Delta u = 0.2 span$ each time all vertices of R_a^u ($a \in \{0, 1\}$) belong to Π.

Such negotiations terminate in < 5 minutes and 20–30 steps. Agreements were found in $> 95\%$ of the instances for which $R_0 \cap R_1 \neq \emptyset$. Fig. 3 shows average time, success rate (i.e., number of negotiations closed successfully / number of negotiations such that $R_0 \cap R_1 \neq \emptyset$), and average quality of the agreement found for each agent (the quality of an agreement D for agent $a \in \{0, 1\}$ is $(u_a(D) - L_a)/(H_a - L_a)$, where H_a and L_a are, respectively, the highest and lowest values of agent a utility in $R_0 \cap R_1$) as a function of the respond policies used (ξ_0 and ξ_1).

It can be seen that moderate respond policies (intermediate values of ξ) lead to very high probabilities ($> 97\%$) of finding an agreement if one exists; moreover, the quality of such agreements for the two agents is similar if their respond policies are similar (fairness). Conversely, if agents use very different values for ξ, the more conceding agent unsurprisingly gets lower utility with the agreement, but negotiations are more often aborted by the other, more demanding, agent.

Structured Instances. We evaluated our system on the 6 scenarios in Table 1a: two scenarios (AB1, AB2) of the Alice vs. Bob example, two scenarios (SU1, SU2) of a negotiation problem regarding the rental of a summerhouse, and two scenarios (EZ1, EZ2) of a variation of the England-Zimbabwe problem of [10] adapted to our domain (real variables and no known bounds for their domains). A description of these negotiation scenarios is omitted for space reasons.

Table 1a shows also some relevant properties of these negotiation scenarios. Column "vars" gives the number of negotiation variables. Columns "polys" and "con" give, respectively, the number of polyhedra and the overall number of



(a) Time (sec)   (b) Success rate   (c) Avg agreement quality

Fig. 3: Experimental results for random instances

Scenario	vars	R_0		R_1		$\frac{vol(R_0 \cap R_1)}{vol(R_0)}$	$\frac{vol(R_0 \cap R_1)}{vol(R_1)}$
		polys	con.	polys	con.		
AB1	2	3	12	3	12	$< 10^{-8}\%$	$< 10^{-8}\%$
AB2	2	3	12	5	20	–	–
SU1	3	3	12	4	16	1.5%	2.0%
SU2	3	3	12	4	16	–	–
EZ1	4	2	8	2	8	0.10%	0.05%
EZ2	4	2	8	2	8	0.15%	0.04%

(a) Properties of negotiation scenarios

Scenario	ξ_0	ξ_1	agr. found	steps	time (sec)	polys
AB1	1	1	Y	20	1.09	321
AB1	0	0	N	24	1.21	450
AB2	1	1	N	20	1.33	466
SU1	0	0	N	417	38.44	20 413
SU1	0.4	0	Y	347	15.22	5679
SU2	1	1	N	513	63.12	29 148
EZ1	0	0	Y	80	1237	3 115 508
EZ2	0	0	Y	92	2836	8 057 011

(b) Experimental results

Table 1: Structured instances

linear constraints defining each agent feasibility region, R_0 and R_1. The two last columns give the ratio of the volume of $R_0 \cap R_1$ (i.e., the volume of the space of the possible agreements) with respect to the volume of the feasibility region of each agent ("–" means that $R_0 \cap R_1$ is empty, hence no agreement is possible).

Table 1b shows some results on the above negotiation scenarios, under different values of the respond policies of each agent (ξ_0 and ξ_1). All instances have been run with k (the maximum number of deals in a proposal) equal to 2. For each instance, column "agr. found" tells whether an agreement has been found (an agreement exists if and only if $R_0 \cap R_1 \neq \emptyset$, see Table 1a), column "steps" gives the number of negotiation steps needed to conclude the negotiation process, column "time" gives the overall negotiation time, and column "polys" gives the overall number of polyhedra computed by PPL during the process. For each negotiation instance, the number of all-SAT instances solved to compute POCs (see formula (1)) is equal to the number of negotiation steps.

Our results show that enforcing NoN is computationally feasible: negotiation processes with hundreds of interaction steps could be performed in minutes, even when NoN enforcement and agents reasoning require the computation of millions of polyhedra and the resolution of hundreds of all-SAT instances.

7 Conclusions

In this paper we defined a new protocol rule, Now or Never (NoN), for bilateral negotiation processes which allows self-motivated competitive agents to *efficiently* carry out multi-variable negotiations with remote untrusted parties, where privacy is a major concern and agents know *nothing* about their opponent. NoN has been explicitly designed as to ensure a continuous progress of the negotiation, thus neutralising malicious or inefficient opponents.

We have also presented a NoN-compliant strategy for an agent that, under mild assumptions on her feasibility region, allows her to derive, in a finite number of steps and *independently* of the behaviour of her opponent, that there is *no hope* to find an agreement. We finally evaluated the computational feasibility of the overall approach on random and structured instances of practical size.

Acknowledgements. This research was founded by the EU 7th Framework Programme under grant agreements n. 317761 (SmartHG) and n. 600773 (PAEON).

References

1. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. of Comp. Progr.*, 72(1–2):3–21, 2008.
2. M. Cadoli. Proposal-based negotiation in convex regions. In *Proc. of CIA 2003*, v. 2782 of *LNCS*, pp. 93–108. Springer, 2003.
3. S. E. Conry, K. Kuwabara, and R. A. Lesser, V. R. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):462–477, 1991.
4. S. Costantini, G. De Gasperis, A. Provetti, and P. Tsintza. A heuristic approach to proposal-based negotiation: with applications in fashion supply chain management. *Math. Problems in Engin.*, 2013. Article ID 896312.
5. P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Intl. J. Robotics & Autonomous Systems*, 24(3–4):159–182, 1998.
6. P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *Artif. Intell.*, 142(2):205–237, 2002.
7. N. R. Jennings, T. J. Norman, P. Faratin, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Appl. Artif. Intell.*, 14(2):145–189, 2000.
8. G. Lai and K. Sycara. A generic framework for automated multi-attribute negotiation. *Group Decision and Negotiation*, 18(2):169–187, 2009.
9. D. Le Berre and A. Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.
10. R. Lin, S. Kraus, D. Tykhonov, K. Hindriks, and C. M. Jonker. Supporting the design of general automated negotiators. In *Proc. of ACAN 2009*, 2009.
11. R. Lin, S. Kraus, J. Wilkenfeld, and J. Barry. Negotiating with bounded rational agents in environments with incomplete information using an automated agent. *Artif. Intell.*, 172(6–7):823–851, 2008.
12. T. Mancini. Negotiation exploiting reasoning by projections. In *Proc. of PAAMS 2009*, Advances in Intelligent and Soft Computing, 2009. Springer.
13. J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiations Among Computers.* The MIT Press, 1994.
14. A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, 1982.
15. A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.
16. K. P. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):446–461, 1991.

An Empirical Perspective on Ten Years of QBF Solving

Paolo Marin[2], Massimo Narizzano[1], Luca Pulina[3], Armando Tacchella[1], and Enrico Giunchiglia[1]

[1] DIBRIS, Università di Genova, Via Opera Pia, 13 – 16145 Genova – Italy
{giunchiglia,narizzano,tacchella}@unige.it
[2] Lehrstuhl für Rechnerarchitektur, Georges-Köhler-Allee 051 – 79110 Freiburg i.B. – Germany marin@informatik.uni-freiburg.de
[3] POLCOMING, Università di Sassari, Viale Mancini n. 5 – 07100 Sassari – Italy
lpulina@uniss.it

Abstract. Twelve years have elapsed since the first QBF evaluation was held as an event linked to SAT conferences. During this period, researchers have strived to propose new algorithms and tools to solve challenging problems, with evaluations periodically trying to assess the current state of the art. In this paper, we present an experimental account of solvers and benchmarks with the aim to understand the progress, if any, in the QBF arena. Unlike typical evaluations, the analysis is not confined to the snapshot of submitted solvers and problems, but rather we consider several tools that were proposed over the last decade, and we run them on different problem sets. The main contribution of our analysis, which is also the message we would like to pass along to the research community is that some faded-to-oblivion techniques turn out to be still quite effective.

1 Introduction

The first non-competitive QBF solvers evaluation (QBFEVAL'03) [3] was held as an associated event of the SAT 2003 conference. If the purpose of QBFEVAL'03 was to assess the state of the art in the relatively young – in the time – QBF reasoning field, the ensuing QBFEVAL series was established with the purpose of measuring the progress in QBF reasoning techniques – see, e.g., [18]. Since the last evaluation, what has been the progress (if any) in the QBF arena? After more than a decade of new solvers being developed and new challenge problems being proposed, we believe that QBFEVAL and, more recently, QBF Gallery [6] events offer a series of snapshots about QBF solving and related aspects, but somehow fail to provide a long-term picture about what has been achieved.

Covering the whole time span of QBFEVAL and QBF Gallery events, our experiments enable us to assess the progress in the QBF field, and put the current state of the art in a historical perspective. In order to achieve this goal, the experimental setup is not confined to a snapshot in time offered by recently proposed systems. In particular, as far as systems are concerned, we consider some *legacy solvers*, i.e., tools that were proposed in the literature, but are not considered in more recents comparative events, e.g., because they are no longer maintained or updated. We call *new solvers* all the other tools that we consider and which are not legacy. In particular, out of 9 solvers considered, the legacy ones are AIGSolve [19], aqme [21], quantor [4], QuBE [8], sKizzo [1], and StruQS [22]. These tools are chosen among winners of at least one category in the past QBFEVAL events, conditioned to their maintenance ending before 2010. The set of new solvers is assembled by including the winners of the last QBF Gallery 2014, namely depqbf [15], ghostq [13] and rareqs [10]. As for problems, we consider two different pools, namely QBF Gallery 2014 Track 1 and QBF Gallery 2014 Track 2. Overall, the problem set is purposefully biased towards more recently submitted instances, in order to (try to) assess legacy solvers on problems that are probably "unseen" to them, i.e., for which their developers did not have a chance to optimize the solver.

The main conclusion that we draw by analyzing the results of our comparison is that the techniques implemented in legacy solvers are far from being outdated. Just to get an idea of what we observe – more details can be found in Section 4 – consider that, if we rank the tools using the number of problems solved, then it turns out that at least two legacy solvers rank among the first three solvers, for all the pools considered. Further evidence in this direction can be obtained considering the "state-of-the-art" (SOTA) solver abstraction, i.e., the ideal system that always fares the best time among the systems in a solver portfolio. If we build "legacy-SOTA" and "new-SOTA" solvers based on the corresponding portfolios of legacy and new solvers, then we observe that legacy-SOTA outperforms new-SOTA – and this remains true even looking at specific subcategories in most cases. While it is difficult to single out the contribution of specific algorithmic techniques by looking at the performances of implemented systems – most of which are closed source – the results we observe strongly suggest that, while new solvers are better engineered than legacy ones, the latter have some combination of techniques that are probably worth taking into account for further developments.

The rest of the paper is structured as follows. In Section 2 we review QBF syntax and semantics. In Section 3 we briefly describe the solvers and the problems used in our experiments. Section 4 presents the results, while in Section 5 we conclude the paper with some final remarks.

2 Preliminaries

In this section we consider the definition of QBFs and their satisfiability as given in the literature of QBF decision procedures (see, e.g., [9, 2, 4]), and we define features describing the structure of QBFs.

Syntax and Semantics A *variable* is an element of a set P of propositional letters and a *literal* is a variable or the negation thereof. We denote with $|l|$ the variable occurring in the literal l, and with $\bar{l}$ the *complement* of l, i.e., $\neg l$ if l is a variable and $|l|$ otherwise. A literal is *positive* if $|l| = l$ and *negative* otherwise. A *clause* C is an n-ary ($n \geq 0$) disjunction of literals such that, for any two distinct disjuncts l, l' in C, it is not the case that $|l| = |l'|$. A *propositional formula* is a k-ary ($k \geq 0$) conjunction of clauses. A *quantified Boolean formula* is an expression of the form

$$Q_1 z_1 \dots Q_n z_n \Phi \tag{1}$$

where, for each $1 \leq i \leq n$, z_i is a variable, Q_i is either an existential quantifier $Q_i = \exists$ or a universal one $Q_i = \forall$, and Φ is a propositional formula in the variables $\{z_1, \dots, z_n\}$. The expression $Q_1 z_1 \dots Q_n z_n$ is the *prefix* and Φ is the *matrix* of (1). A literal l is *existential* if $|l| = z_i$ for some $1 \leq i \leq n$ and $\exists z_i$ belongs to the prefix of (1), and it is *universal* otherwise.

The semantics of a QBF φ can be defined recursively as follows. A QBF clause is *contradictory* exactly when it does not contain existential literals. If the matrix of φ contains a contradictory clause then φ is false. If the matrix of φ has no conjuncts then φ is true. If $\varphi = Qz\psi$ is a QBF and l is a literal, we define φ_l as the QBF obtained from ψ by removing all the conjuncts in which l occurs and removing $\bar{l}$ from the others. Then we have two cases. If φ is $\exists z\psi$, then φ is true exactly when φ_z or $\varphi_{\neg z}$ are true. If φ is $\forall z\psi$, then φ is true exactly when φ_z and $\varphi_{\neg z}$ are true. The QBF satisfiability problem (QSAT) is to decide whether a given formula is true or false. It is easy to see that if φ is a QBF without universal quantifiers, solving QSAT is the same as solving propositional satisfiability (SAT).

Representing QBFs To correlate the structure of QBFs with the performances of solvers, we extract representative features from QBFs — see, e.g., [21]. A first class is given by syntactic features:

- c, total number of clauses; c_1, c_2, c_3 total number of clauses with 1, 2 and more than two existential literals, respectively; c_h, c_{dh} total number of Horn and dual-Horn clauses, respectively;
- v, total number of variables; $v_\exists$, $v_\forall$, total number of existential and universal variables, respectively; l_{tot}, total number of literals; vs, $vs_\exists$, $vs_\forall$, distribution of the number of variables per quantifier set, considering all the variables, and focusing on existential and universal variables, respectively; s, $s_\exists$, $s_\forall$, number of total, existential and universal, quantifier sets;
- l, distribution of the number of literals in each clause; l_+, l_-, $l_\exists$, $l_{\exists+}$, $l_{\exists-}$, $l_\forall$, $l_{\forall+}$, $l_{\forall-}$, distribution of the number of positive, negative, existential, positive existential, negative existential, universal, positive universal, negative universal number of literals in each clauses, respectively.
- r, distribution of the number of variable occurrences r_+, r_-, $r_\exists$, $r_{\exists+}$, $r_{\exists-}$, $r_\forall$, $r_{\forall+}$, $r_{\forall-}$, distribution of the number of positive, negative, existential, positive existential, negative existential, universal, positive universal, negative universal variable occurrences, respectively.

We also take into account the following combined features:

- $\frac{c}{v}$, the classic clauses-to-variables ratio, and for each $x \in \{l, r\}$ the following ratios (on mean values):
  - $\frac{x_+}{x}$, $\frac{x_-}{x}$, $\frac{x_+}{x_-}$, balance ratios;
  - $\frac{x_\exists}{x}$, $\frac{x_{\exists+}}{x}$, $\frac{x_{\exists-}}{x}$, $\frac{x_{\exists+}}{x_\exists}$, $\frac{x_{\exists-}}{x_\exists}$, $\frac{x_{\exists+}}{x_{\exists-}}$, $\frac{x_{\exists+}}{x_+}$, $\frac{x_{\exists-}}{x_-}$, balance ratios (existential part);
  - $\frac{x_\forall}{x}$, $\frac{x_{\forall+}}{x}$, $\frac{x_{\forall-}}{x}$, $\frac{x_{\forall+}}{x_+}$, $\frac{x_{\forall-}}{x_-}$, $\frac{x_{\forall+}}{x_\forall}$, $\frac{x_{\forall-}}{x_\forall}$, $\frac{x_{\forall+}}{x_{\forall-}}$, balance ratios (universal part);
- $\frac{c_1}{c}$, $\frac{c_2}{c}$, $\frac{c_3}{c}$, $\frac{c_h}{c}$, $\frac{c_{dh}}{c}$, $\frac{c_h}{c_{dh}}$, i.e., balance ratios between different kinds of clauses.

A second class of features is computed on graph models of QBFs. From previous related work on SAT, see, e.g. [17], we borrow variable graphs (VG) and the clause graphs (CG). The former has a node for each variable and an edge between variables that occur together in at least one clause, while the latter has nodes representing clauses and an edge between two clauses whenever they share a negated literal. For each graph, we consider the average value on their node degree. Finally, we also consider a treewidth measure tw_p which accounts for the treewidth of the VG adjusted to keep into account that only elimination orders compatible to the prefix p are viable — see [20, 23] for details, and also for extensive empirical evidence about the correlation of tw_p with hardness of QBFs.

3 Setup

In this section we present solvers and problems that we selected for our analysis. As for solvers, we consider systems participating to QBF Gallery 2014[1] as well as solvers participating to past QBFEVAL editions. Considering the former, we choose the winners of Track 1 and Track 2 [12] which are shortly described in the following.

DEPQBF (v. 3.0.4) [15] is a search-based solver performing non-chronological backtracking from conflicts and solutions; DEPQBF can select branching variables without following the prefix order by leveraging a compact representation of the dependencies among variables.

GHOSTQ (v. qdimacs-gal-2014) [13] is a non-prenex DPLL-based solver which makes use of auxiliary variables to force necessary assignments, i.e., to force a value to an existential (resp. universal) variable if the opposite value directly makes the formula evaluate to false (resp. true). Additionally, it features a CEGAR-based learning to further prune the search space when the last decision literal is existential (resp. universal) and a conflict (resp. solution) is detected.

RAREQS (v. 1.1) [10] is a counterexample guided abstraction refinement (CEGAR) based solver which performs a kind of resolution and expansion procedure but in a depth-first way, i.e., by expanding first only one value of a variable, and learns abstractions of the local partial solutions to refine the global solution.

[1] http://qbf.satisfiability.org/gallery

We did not consider the system HIQQER [12] because we could not find a version available for download. In the remainder of the paper, we refer to this pool of solvers as S-NEW.

Solvers participating to past editions of QBFEVAL – to which we refer as S-LEGACY from now on – are described in the following.

AIGSOLVE [19] uses And-Inverter Graphs (AIGs) as the main data structure, and AIG-based operations to reason about the input formula. The solver includes preliminary phases devoted to simplification, structure extraction and early quantification of the input formula.

AQME [21] is a multi-engine solver, i.e., a tool using Machine Learning techniques to select among its reasoning engines the one which is more likely to yield optimal results. The reasoning engines of AQME are a subset of those submitted to QBFEVAL'06, namely 2CLSQ, QUANTOR, QUBE, SKIZZO, and SSOLVE. Engine selection is performed according to the adaptive strategy described in [21].

QUANTOR [4] is based on Q-resolution (to eliminate existential variables) and Shannon expansion (to eliminate universal variables), plus a number of features, such as equivalence reasoning, subsumption checking, pure literal detection, unit propagation, and also a scheduler for the elimination step.

QUBE [8] is a solver that first applies, among other simplification techniques, deep equivalence reasoning and removes variables by Q-Resolution. Then, it uses a search-based decision procedure that performs monotone and “don't care” literal propagation, non-chronological backtracking from conflicts and solutions, in which it produces and removes less clauses/terms made tautological by blocking universal/existential literals than its predecessor.

SKIZZO [1] is a reasoning engine for QBF featuring several techniques, including search, resolution and skolemization.

STRUQS [22] main feature is a dynamic combination of search – with solution- and conflict-backjumping – and variable-elimination. The key point in this approach is to implicitly leverage graph abstractions of QBFs to yield structural features which support an effective decision between search and variable elimination.

We included AIGSOLVE because it is the only system employing AIG-based operations to reason on input QBF. We involved AQME for its multi-engine architecture; as a by-product, it can return an approximated picture of state-of-the-art QBF solvers back in 2006, so it can be used as “yardstick” to assess improvements. QUANTOR, QUBE, and SKIZZO implement key QBF solution techniques, namely resolution and expansion, DPLL-search, and Skolemization, respectively. Finally, we included STRUQS because it represents the first — and, to the best of our knowledge, the only — attempt to combine dynamically very different solution techniques. Almost all the S-LEGACY solvers also collected accolades in past QBF evaluations. AIGSOLVE was the winner of the QBFEVAL'10 small hard track, while AQME was the system able to solve the highest number of formulas in QBFEVAL'07, '08, and in the main track of QBFEVAL'10. QUANTOR was the winner of QBFEVAL'04, while QuBE won the 2QBF track of QBFEVAL'10. Finally, sKizzo has been the winner of QBFEVAL'05 and '07.

We evaluate the above mentioned systems on different pools of problem instances. The syntax of the instances is prenex-CNF using the `qdimacs` 1.1 format. The problem pools we consider are briefly outlined in the following.

- The formulas included in QBF Gallery 2014 Track 1. These are 276 instances collectively denoted as QBFG-T1.
- The formulas included in QBF Gallery 2014 Track 2. These are collectively denoted as QBFG-T2.

The pool QBFG-T2 includes formulas coming from six different families, namely:

`bomb` and `dungeon` [14] are encodings of conformant planning problems with optimal length and uncertainty of the initial state.

`complexity` [11] result from a QBF encoding of automatic reduction between decision problems. The original problem is undecidable in general, but it can be reduced to Σ_2^p if the dimension of the reduction is fixed and given, and the size of the inputs is bounded.

`hardness` [16] Black-Box bounded model checking instances for an incomplete parametrized arbiter of a bus system.

`planning` [5] This instance set include different planning problems encoded into QBF using two different strategies: the first one is based on the iterative squaring formulation, and the second one relies on a more compact tree-like encoding.

`testing` [24] The solutions to these problems are test patterns for sequential circuits coming from ISCAS 89 and ITC 99 benchmarks having a maximum amount of inputs set to don't care.

4 Experimental Analysis

In this section we report and analyze the results of our empirical evaluation. All the experiments ran on a cluster of Intel Xeon E3-1245 PCs at 3.30 GHz equipped with 64 bit Ubuntu 12.04. All solvers were limited to 600 seconds of CPU time and to 4GB of memory.

4.1 QBF Gallery 2014 formulas – Track 1

The aim of our first experiment is to evaluate the selected solvers in the QBFG-T1 pool of instances. In Table 1 we report the raw results of such evaluation. Looking at the results, we can see that only 6 solvers out of 9 were able to solve at least 25% of the test set. If we rank solvers according to the number of problems solved within the time limit, then the best system is AIGSolve, which can solve about 42% of the test set, followed by QuBE and AQME. To find the best solver in S-NEW, namely GHOSTQ, we must go down to the fourth position. GHOSTQ performs only slightly worse than AQME, and it tops at 33% of the

Solver	Total		True		False		Unique	
	#	Time	#	Time	#	Time	#	Time
AIGSolve	116	5333.01	56	2177.45	60	3155.56	22	1458.26
QuBE	106	8764.73	53	3997.78	53	4766.95	8	1195.58
aqme	97	3287.20	39	1098.00	58	2189.20	–	–
ghostq	91	4814.73	48	2912.38	43	1902.17	4	158.97
depqbf	88	2388.32	39	1163.15	49	1225.17	5	454.77
rareqs	79	2588.64	32	1593.25	47	995.39	6	787.33
sKizzo	51	948.81	18	556.76	33	392.06	–	–
quantor	50	1498.37	28	911.72	22	586.65	2	161.67
StruQS	43	6092.64	31	4052.98	12	2039.66	1	16.53

Table 1. Runtime of solvers on qbfg-t1. For each solver, the table reports its name (column "Solver"), the total number of instances solved and the cumulative time to solve them (columns "#" and "Time", group "Total"), the number of instances found satisfiable and the time to solve them (columns "#" and "Time", group "True"), the number of instances found unsatisfiable and the time to solve them (columns "#" and "Time", group "False"), and, finally, the number of instances uniquely solved and the time to solve them (columns "#" and "Time", group "Unique"); a "–" (dash) means that the solver did not solve any instance. The table is sorted in descending order according to the number of instances solved, and, in case of a tie, in ascending order according to the cumulative time taken to solve them.

test set. This result is relevant for our case in point, particularly if we consider that both AIGSolve and QuBE are systems dating back to 2010, while aqme combines solvers dating back to QBFEVAL 2006. Finally, despite quantor and StruQS were not able to solve more than 20% of qbfg-t1, still they were the only ones able to solve some instances — 2 and 1, respectively.

If we consider the structure of the instances comprised in qbfg-t1, then we can observe several structural differences between those solved by at least one solver and those that remained unsolved. For instance, if we focus on the formula size in terms of variables v and clauses c, then we can see that unsolved instances feature, on average, higher values of both parameters, i.e., they are somewhat larger. Looking at the median values $\hat{v}$ and $\hat{c}$ of the parameters v and c, we can see that $\hat{v} = 3412$ if the population is restricted to solved instances, whereas $\hat{v} = 10188$ on the population of unsolved ones. A similar picture holds for c, with $\hat{c} = 14818$ and $\hat{c} = 57130$ for solved and unsolved instances, respectively. As expected, tw_p is also indicative of this spread, since $\hat{tw}_p = 486$ for solved instances, whereas $\hat{tw}_p = 1102$ for unsolved ones.

Another perspective about the results of Table 1 can be obtained by resorting to the *state-of-the-art solver* abstraction (sota in the following), i.e., the ideal solver that always fares the best time among all the solvers in a portfolio. In this case, sota was able to cope with about 73% of qbfg-t1 (202 formulas). What is more relevant is that all the systems contributed to its composition. In particular, the main contributors — in percentage — were AIGSolve, depqbf,

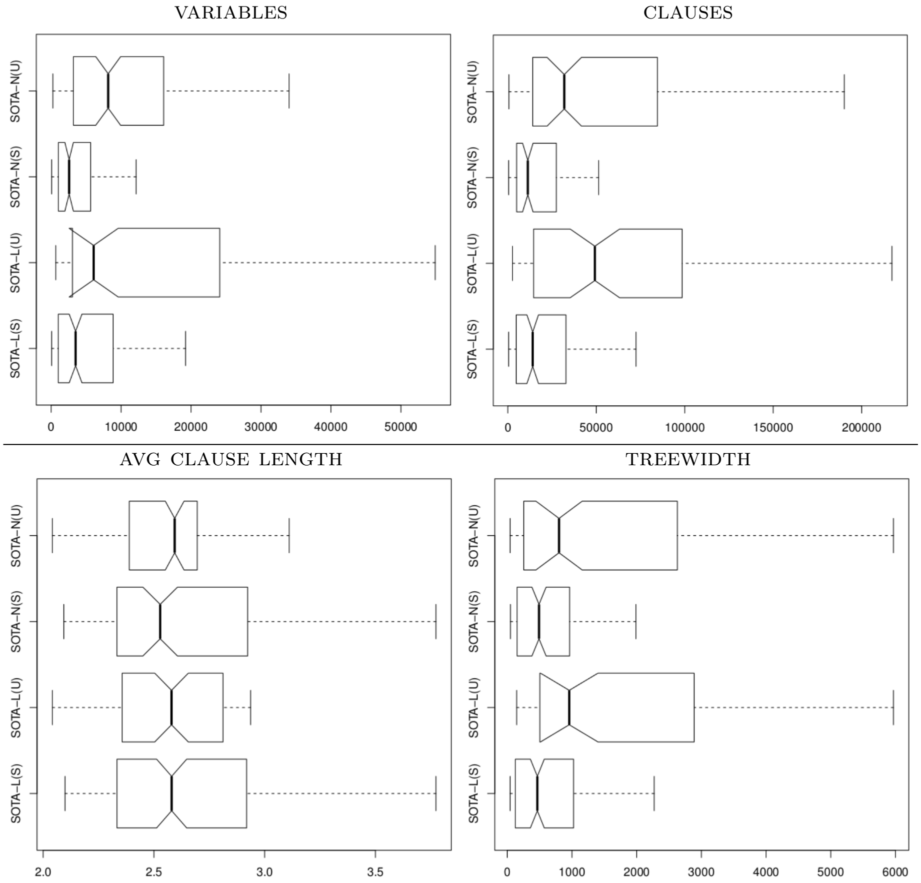


Fig. 1. Box-plots of different features distributions related to QBFs comprised in QBFG-T1. Features are v and c (top-left and top-right, respectively), l (bottom left), and tw_p (bottom right). Each distribution in the plots is labeled as follows: "SOTA-L" and "SOTA-N" are placeholders for SOTA-LEGACY and SOTA-NEW, respectively, while "S" and "U" – in parentheses – stand for "solved" and "unsolved". For each plot, we show a box-and-whiskers diagram representing the median (bold line), the first and third quartile (bottom and top edges of the box), the minimum and maximum (whiskers at the top and the bottom) of a distribution. An approximated 95% confidence interval for the difference in the two medians is represented by the notches cut in the boxes: if the notches of two plots do not overlap, this is strong evidence that the two medians differ. Finally, in the y-axes of each plot are reported the values of the related features.

and RAREQS with 22%, 20%, and 17%, respectively. Notice that 2 out of 3 of the main contributors are indeed in S-NEW.

With the aid of the SOTA abstraction we can also compare the overall performances of solvers in S-LEGACY vs. those in S-NEW. In order to do that, we compute two abstractions, namely SOTA-LEGACY — considering only legacy systems

— and SOTA-NEW— considering only new solvers. The rationale of this analysis is twofold: on one hand, we want to evaluate the advancement of the state of the art with respect to legacy systems (and related solving techniques); on the other, we want to look for patterns, expressed by means of features, enabling us to spot differences in the type of QBFs solved by old and new systems. As far as advancing the state of the art is concerned, we report that SOTA-LEGACY solves 185 formulas — about 92% of those solved by SOTA — while SOTA-NEW tops at 70% (142 instances). In view of these results, and considering that most of the formulas in QBFG-T1 where not available at the time in which the solvers in S-LEGACY were developed, there does not seem to be a stark advancement in solvers' abilities from S-LEGACY to S-NEW.

As for the nature of the instances solved by legacy vs. new solvers, we can try to observe differences in the structure of QBFs solved by solvers in SOTA-LEGACY and solvers in SOTA-NEW. In Figure 1 we present the distributions of four features across four different populations obtained by combining SOTA-LEGACY, SOTA-NEW with solved and unsolved formulas. In the figure, we can see that the parameter l (average clause length) is not significantly different among the various classes of problems — all the notches overlap. If we consider v (number of variables) then we see that for SOTA-NEW the value of $\hat{v}$ is significantly different between solved and unsolved instances, while the same is not true for SOTA-LEGACY. Therefore, it seems that the sheer number of variables matters most for solvers in SOTA-NEW. However, also notice that there is no significant difference between SOTA-LEGACY and SOTA-NEW when considering (un)solved formulas. As for c (number of clauses) both SOTA-LEGACY and SOTA-NEW are sensitive to this parameter: higher values of c imply harder formulas. Also in this case, no significant difference can be spotted when considering SOTA-LEGACY and SOTA-NEW on (un)solved formulas. Finally, looking at the distributions of tw_p, we can see that its median value is not a significant hardness predictor for solvers in SOTA-NEW, whereas it is a hardness predictor for solvers in SOTA-LEGACY, but there are no differences when considering (un)solved formulas. Overall, we can conclude that no clear pattern emerges that could help to differentiate (un)solved formulas between solvers in SOTA-NEW and SOTA-LEGACY, at least looking at the parameters shown in Figure 1.

4.2 QBF Gallery 2014 formulas – Track 2

Our next experiment aims at assessing solvers on the pool QBFG-T2. Before delving into the analysis, we wish to point out that there are structural differences between the formulas in QBFG-T2 and those in QBFG-T1. On average, they are characterized by a smaller number of median variables $\hat{v}$ (3374 vs. 4708), but a considerably larger number of median clauses $\hat{c}$ (29492 vs. 17397). Formulas in QBFG-T2 are also characterized by a relatively small value of universal variables since $\frac{\hat{v}_\forall}{\hat{v}} = 0.006$ in the case of QBFG-T2, while the same ratio is 0.02 in the case of QBFG-T1. Finally, we report that QBFG-T1 formulas usually have a higher value of average clause length since $\hat{l} = 2.58$, whereas the same value is 2.37 for QBFG-T2.

Family	Solver	Total		True		False		Unique	
		#	Time	#	Time	#	Time	#	Time
bomb (132)	AIGSolve	83	1003.23	40	165.20	43	838.03	–	–
	rareqs	83	1420.61	34	165.41	49	1255.19	6	1094.71
	quantor	82	923.25	53	217.92	29	705.33	–	–
	aqme	80	674.38	53	345.02	27	329.36	–	–
	depqbf	67	2410.16	40	1693.36	27	716.79	–	–
	sKizzo	57	609.41	31	2.39	26	607.03	–	–
	ghostq	56	532.47	29	42.66	27	489.81	–	–
	QuBE	47	1168.86	23	470.47	24	698.39	–	–
	StruQS	36	1051.46	19	813.58	17	237.88	–	–
complexity (104)	rareqs	75	1559.65	29	466.77	46	1092.88	15	1148.51
	depqbf	49	1553.73	22	1086.35	27	467.38	–	–
	ghostq	42	1791.86	11	499.21	27	467.38	–	–
	QuBE	39	1273.95	19	277.87	20	996.09	–	–
	aqme	33	528.28	15	188.76	18	339.52	–	–
	quantor	26	170.44	11	11.29	15	159.14	–	–
	StruQS	21	1855.53	13	1677.81	8	177.72	–	–
	AIGSolve	15	70.26	7	12.24	8	58.02	–	–
	sKizzo	9	316.60	4	315.82	5	0.78	–	–
dungeon (107)	quantor	104	525.30	18	54.81	86	470.48	–	–
	aqme	104	1121.43	18	86.11	86	1035.32	–	–
	AIGSolve	87	1220.22	17	417.12	70	803.10	–	–
	rareqs	57	1870.73	18	54.89	39	1815.85	–	–
	depqbf	44	535.22	18	300.44	26	234.77	–	–
	QuBE	34	1429.60	7	212.89	27	1216.71	–	–
	ghostq	7	385.11	4	4.62	3	380.49	–	–
	sKizzo	2	0.99	–	–	2	0.99	–	–
	StruQS	1	21.96	1	21.96	–	–	–	–
hardness (114)	StruQS	88	7826.42	1	372.74	87	7453.68	12	3033.19
	QuBE	76	1346.11	–	–	76	1346.11	2	328.75
	ghostq	51	2649.30	2	239.56	49	2409.74	1	224.22
	aqme	50	265.14	–	–	50	265.14	–	–
	rareqs	14	1431.05	–	–	14	1431.05	–	–
	AIGSolve	12	2038.84	–	–	12	2038.84	–	–
	depqbf	8	617.99	–	–	8	617.99	–	–
	quantor	–	–	–	–	–	–	–	–
	sKizzo	–	–	–	–	–	–	–	–
planning (147)	AIGSolve	147	2371.36	38	114.02	109	2257.34	10	861.70
	rareqs	137	1093.01	38	125.66	99	967.35	–	–
	quantor	131	6750.13	37	122.68	94	6627.44	–	–
	aqme	123	9263.25	37	464.97	86	8798.28	–	–
	sKizzo	74	71.57	34	24.02	40	47.55	–	–
	depqbf	57	5134.24	29	1876.90	28	3257.34	–	–
	QuBE	14	1270.35	12	743.61	2	526.74	–	–
	ghostq	11	2155.26	8	1420.71	3	734.55	–	–
	StruQS	4	1229.67	4	1229.67	–	–	–	–
testing (131)	aqme	71	2675.64	64	2339.68	7	335.95	1	3.00
	StruQS	65	1770.09	63	1488.09	2	282.00	4	236.18
	depqbf	57	692.38	46	672.96	11	19.42	2	359.15
	AIGSolve	51	4194.44	46	4163.65	5	30.79	2	11.88
	QuBE	41	765.08	31	734.85	10	30.23	1	1.24
	rareqs	34	428.00	22	317.04	12	110.95	1	0.53
	ghostq	32	269.13	29	66.10	3	203.03	–	–
	quantor	26	121.15	25	110.52	1	10.63	–	–
	sKizzo	1	0.02	1	0.02	–	–	–	–

Table 2. Performances of QBF solvers on qbfg-t2: The table is split in six horizontal parts, one for each family. The first column contains families names, as well as its total amount of instances. The rest of the table is organized as Table 1.

In Table 2, we show the results of our experiments on QBFG-T2. The formulas in `bomb`, when compared to the whole QBFG-T2 formulas, are characterized by higher median values of $\frac{l_-}{l}$ (0.96 vs 0.84), $\frac{c}{v}$ (13.61 vs 8.74), and tw_p (914 vs 758). On this subcategory, the best systems are AIGSolve, RAREQS, and QUANTOR, which are the only ones able to solve more than 60% of the total. Also in this case, two of the top three performers are solvers in S-LEGACY. However, we should also point out that RAREQS is the only system able to solve instances uniquely. Overall, it seems that the solvers which are not purely search-based are also the most effective ones in this subcategory. This difference cuts across the separation between S-LEGACY and S-NEW, and it could be due to the fact that these formulas are relatively easy to expand into SAT instances, so solvers featuring this technique, e.g., QUANTOR and RAREQS, handle them more effectively. Indeed, if we consider the SOTA abstraction, its major contributors are QUANTOR and RAREQS, with 41 and 38 formulas, respectively. Overall, SOTA is able to solve 77% of the total (102 instances out of 132). In spite of the very good performances of RAREQS, still SOTA-LEGACY solved 96 instances, while SOTA-NEW 89, thus confirming the picture that we observed in QBFG-T1.

Regarding the results on `complexity`, looking at Table 2 we can see that the best solvers are all comprised in S-NEW. Noticeably, this is the only subcategory of QBFG-T2 and the only case throughout our experimental analysis in which this is true. RAREQS, DEPQBF, and GHOSTQ are able to solve 75, 49, and 42 instances, respectively. In particular, RAREQS solves 15 of them uniquely. Looking at the structure of QBFs, we can see that `complexity` instances are smaller than `bomb` ones: the median values of c, v, and l are 1101, 2533, and 6601, respectively. Moreover, with respect to `bomb`, we also report a smaller values of $\frac{\hat{v}_\forall}{\hat{v}}$, and of median clause-to-variable ratio $\frac{c}{v}$. On the other hand, the parameter $\hat{l}$ on these formulas is 2.66, higher than QBFG-T2 (2.37) and `bomb` (2.07). Since DEPQBF and GHOSTQ do not perform very well on `bomb` and also on other subcategories in QBFG-T2, we conjecture that (i) relatively small instances with (ii) relatively small number of universally quantified variables even with (iii) relatively long clauses, could correlate with positive performances of DEPQBF and GHOSTQ. Considering the SOTA abstraction, we report that it solves the same number of formulas solved by the best solver (RAREQS). Unsurprisingly, in this case SOTA-NEW outperforms SOTA-LEGACY — 75 and 44 solved instances, respectively.

Considering `dungeon`, we can see that the three best solvers are comprised in S-LEGACY. QUANTOR and AQME solved 97% of `dungeon`, while AIGSolve topping at about 81%. The structure of `dungeon` is characterized by large values of $\hat{v}$, $\hat{c}$, and $\hat{l}$ (27781, 128155, and 265184, respectively). On the other hand, we report small values of $\hat{l}$ (1.99) and $\hat{v_\forall}$ (5). Moreover, it is worth noticing that `dungeon` formulas have a large amount of c_1 and c_h (number of unary and Horn clauses, respectively) with respect to the whole QBFs in QBFG-T2. The value of $\hat{c_1}$ related to `dungeon` is 20299, while the one reported for QBFG-T2 is 3. Considering $\hat{c}_h$, the values in `dungeon` and QBFG-T2 are 125611 and 23277, respectively. Given, e.g., the large number of unary and Horn clauses, these formulas should not be particularly challenging in general. Despite that, looking at the result we can see that otherwise effective solvers such as GHOSTQ solved only 6% of the total. This fact makes us conjecture that for this family sheer size becomes an issue for some solvers. Finally, we report that SOTA can solve all but one formula (106 solved out of 107) and, in this case, SOTA-LEGACY outperforms SOTA-NEW (106 and 57 solved instances, respectively).

Considering `hardness`, looking at Table 2 we can see that the best system is StruQS with 88 solved formulas, followed by QuBE and GHOSTQ with 76 and 51 solved instances, respectively. This result is quite surprising because, considering the results described so far, StruQS always ranks among the worst three solvers. To investigate this phenomenon, we analyzed the structure of the instances comprised in `hardness`. First, we report that, on one hand, both $\hat{v}$ and $\hat{c}$ are relatively small (2191 and 7793, respectively); on the other, we can report for `hardness` the highest value of several features, such as $\hat{l}$ (9.80), $\frac{\hat{v_\forall}}{\hat{v}}$ (in percentage, 5%), and the number of quantified sets $\hat{s}$ (26, against a value of 3 reported for QBFG-T2). This can partially explain the performance of StruQS because its hybrid resolution-search algorithm works best with small formulas having many quantifier alternations. Finally, we report that SOTA was able to solve 91 instances, and its best contributors – in percentage – are QuBE, StruQS, and GHOSTQ, with 65%, 16%, and 13% of the total, respectively. Also in this case, SOTA-LEGACY outperformed SOTA-NEW (90 and 51 solved instances, respectively).

Concerning the results on `planning`, we can see from Table 2 the best solver is AIGSolve, able to deal with all the instances in the family. It is followed by RAREQS and QUANTOR, that solved 137 and 131 instances, respectively. The picture seems to be very similar to `bomb` and, indeed we can report that this family is characterized by a low value of $\hat{v}$ (1947 vs. 3374 of QBFG-T2), but large values of $\hat{l_{tot}}$ (326955 vs 96532) and $\hat{c}$ (112826 vs 29492). These data also implies that `planning` has the largest value of the median clause to variable ratio. Finally, we report that variables in `planning` are highly connected: the median value of the VG node degree is 170.05, while the same value in QBFG-T2 is 21.61. As a final comment, we report that the performances of SOTA-LEGACY and SOTA-NEW are quite close in this case, with 147 and 137 instances solved, respectively.

To conclude, looking at the results on `testing`, we can see that AQME is the best solver, dealing with about 54% of the instances. It is worth noticing that AQME solved 13% of the instances running sSolve [7], a system dating back to year 2000. Second and third are StruQS and DEPQBF, solving about 50% and 43%, respectively. Values of dimensional features of these formulas are quite similar to `bomb`, with the noticeable exception of the total amount of universal variables (more than 2% of the total), that makes the family more similar to `hardness`, which may also can explain the good performances of StruQS. As a final consideration, we report that SOTA solved 69% of the total, and its major contributors is DEPQBF (53%). Notice that also in this case SOTA-LEGACY outperforms SOTA-NEW (85 and 65 solved instances, respectively).

5 Conclusions

In the paper we have shown the results of a massive evaluation of QBF solvers and benchmarks. The picture that we have obtained is significant both because it is the first historical perspective on QBF solving technologies, and because of the results that emerged clearly from the analysis. In particular, we have shown that recently proposed solvers might benefit from some techniques implemented in legacy ones which defy aging. Indeed, new solvers seems to be fairly well engineered – the majority of the overall SOTA solver is made by new systems – and they made a relevant contribution to the QBF field, as witnessed by the fact that they are most often among the ones solving a formula uniquely. However, by comparing the SOTA-LEGACY and SOTA-NEW abstractions we have also shown that legacy systems still outperform the new ones in many problem categories. Therefore, we believe that it would be interesting to blend new techniques, e.g., CEGAR or dependency schemas, with legacy ones – modulo the inevitable engineering challenges that might arise – in order to really push forward the state of the art in the QBF arena. A contribution to the development and optimization of such blended solvers might come, e.g., from the significant number of problems that emerged as challenging throughout our evaluation.

References

1. M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Eleventh International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2004)*, volume 3452 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.
2. M. Benedetti. sKizzo: a Suite to Evaluate and Certify QBFs. In *20th Int.l. Conference on Automated Deduction*, volume 3632 of *LNCS*, pages 369–376. Springer Verlag, 2005.
3. D. L. Berre, L. Simon, and A. Tacchella. Challenges in the QBF arena: the SAT'03 evaluation of QBF solvers. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*, pages 468–485. Springer Verlag, 2003.
4. A. Biere. Resolve and Expand. In *Seventh Intl. Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 of *LNCS*, pages 59–70. Springer Verlag, 2005.
5. M. Cashmore, M. Fox, and E. Giunchiglia. Partially grounded planning as quantified boolean formula. In D. Borrajo, S. Kambhampati, A. Oddi, and S. Fratini, editors, *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*. AAAI, 2013.
6. A. V. G. F. Lonsing, M. Seidl. QBF gallery 2013, 2013. `http://www.kr.tuwien.ac.at/events/qbfgallery2013/`.
7. R. Feldmann, B. Monien, and S. Schamberger. A distributed algorithm to evaluate quantified boolean formulae. In *Proceedings of the Seventeenth National Conference in Artificial Intelligence (AAAI'00)*, pages 285–290. AAAI Press / The MIT Press, 2000.

8. E. Giunchiglia, P. Marin, and M. Narizzano. Qube7.0. *JSAT*, 7(2-3):83–88, 2010.
9. E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause-Term Resolution and Learning in Quantified Boolean Logic Satisfiability. *Artificial Intelligence Research*, 26:371–416, 2006.
10. M. Janota, W. Klieber, J. Marques-Silva, and E. Clarke. Solving QBF with counterexample guided refinement. In *Theory and Applications of Satisfiability Testing–SAT 2012*, pages 114–128. Springer Berlin Heidelberg, 2012.
11. C. Jordan and L. Kaiser. Experiments with reduction finding. In M. Jarvisalo and A. Van Gelder, editors, *Theory and Applications of Satisfiability Testing SAT 2013*, volume 7962 of *Lecture Notes in Computer Science*, pages 192–207. Springer Berlin Heidelberg, 2013.
12. C. Jordan and M. Seidl. The QBF Gallery 2014, 2014.
13. W. Klieber, S. Sapra, S. Gao, and E. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *Theory and Applications of Satisfiability Testing–SAT 2010*, pages 128–142. Springer, 2010.
14. M. Kronegger, A. Pfandler, and R. Pichler. Conformant planning as a benchmark for QBF solvers. In *Intl. Workshop on Quantified Boolean Formulas (QBF 2013)*, page 15, 2013.
15. F. Lonsing and A. Biere. Depqbf: A dependency-aware QBF solver. *JSAT*, 7(2-3):71–76, 2010.
16. C. Miller, C. Scholl, and B. Becker. Proving QBF-hardness in Bounded Model Checking for Incomplete Designs. In *14th International Workshop on Microprocessor Test and Verification, MTV 2013, Austin, TX, USA, December 11-13, 2013*, pages 23–28. IEEE Computer Society, 2013.
17. E. Nudelman, K. Leyton-Brown, A. Devkar, Y. Shoham, and H. Hoos. SATzilla: An Algorithm Portfolio for SAT. In *In Seventh International Conference on Theory and Applications of Satisfiability Testing, SAT 2004 Competition: Solver Descriptions*, pages 13–14, 2004.
18. C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. The seventh QBF solvers evaluation (QBFEVAL'10). In *Theory and Applications of Satisfiability Testing–SAT 2010*, pages 237–250. Springer Berlin Heidelberg, 2010.
19. F. Pigorsch and C. Scholl. An AIG-based QBF-solver using SAT for preprocessing. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 170–175. IEEE, 2010.
20. L. Pulina and A. Tacchella. Treewidth: a useful marker of empirical hardness in quantified Boolean logic encodings. In *15th Int.l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 5330 of *LNCS*, pages 528–542. Springer, 2008.
21. L. Pulina and A. Tacchella. A self-adaptive multi-engine solver for quantified Boolean formulas. *Constraints*, 14(1):80–116, 2009.
22. L. Pulina and A. Tacchella. A structural approach to reasoning with quantified Boolean formulas. In *21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 596–602, 2009.
23. L. Pulina and A. Tacchella. An empirical study of QBF encodings: from treewidth estimation to useful preprocessing. *Fundamenta Informaticae*, 102(3):391–427, 2010.
24. M. Sauer, S. Reimer, I. Polian, T. Schubert, and B. Becker. Provably optimal test cube generation using quantified boolean formula solving. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 533–539, Jan 2013.

Modeling Abduction over Acyclic First-Order Logic Horn Theories in Answer Set Programming: Preliminary Experiments*

Peter Schüller

Computer Engineering Department, Faculty of Engineering
Marmara University, Turkey
peter.schuller@marmara.edu.tr

Abstract. We describe encodings in Answer Set Programming for abductive reasoning in First Order Logic in acyclic Horn theories in the presence of value invention and in the absence of Unique Names Assumption. We perform experiments using the Accel benchmark for abductive plan recognition in natural language processing. Results show, that our encodings cannot compete with state-of-the-art realizations of First Order Logic abduction, mainly due to large groundings. We analyze reasons for this bad performance and outline potential future improvements.

1 Introduction

Abduction [29] is reasoning to the best explanation, which is an important topic in diverse areas such as diagnosis, planning, and natural language processing.

We experiment with the Accel benchmark [27] for abduction in acyclic First Order Logic (FOL) Horn theories and model this problem in Answer Set Programming (ASP) [25]. Existing methods for solving Accel use backward-chaining search [28], Markov Logic [22, 2], or Integer Linear Programming (ILP) [19].

In this work we realize abduction for Accel in the declarative mechanism of Answer Set Programming. Our motivated is twofold: (a) it will allow for adding complex constraints and optimization criteria more flexibly than in search-based realizations of abduction, moreover (b) it can provide insights for managing other problems with prohibitively large groundings in ASP.

Tackling this problem in ASP seems possible, because a recent solution for Accel is based on first creating an ILP theory (in Java) and then solving it [19].

Yet there are two major challenges when realizing FOL abduction in ASP.

- The Unique Names Assumption (UNA) is not applicable in FOL in general. In particular the Accel benchmark requires a partial UNA that holds only for sort names. In ASP, on the contrary, UNA holds for all ground terms.
- Backward-chaining requires instantiation of body variables with new constant terms. In ASP this is equivalent to existential variables in rule heads,

* This work has been supported by Scientific and Technological Research Council of Turkey (TUBITAK) Grant 114E777.

i.e., variables that are absent in the rule body which makes the rule unsafe.

In ASP rules must be safe, therefore we need to encode value invention. Note that value invention makes FOL abduction undecidable in general. The Accel knowledge base is an acyclic theory,[1] therefore undecidability is not an issue and we can realize value invention in ASP using Skolemization.

Partial UNA has the effect that we must encode equivalence classes between terms and effects of these equivalences in ASP. Sort names make the problem of finding the smallest set of abducibles that explains a goal nontrivial (without sort names the smallest explanation is those where all terms are equivalent).

Both value invention and partial UNA make an ASP solution tricky, in particular the size of the instantiated program can become problematic easily.

In this study we approach these challenges and analyze problems that become apparent. We describe three encodings which use different rewritings of the Accel knowledge base and an observation that should be explained into ASP:

- BackCh models a backward-chaining algorithm and the notion of 'factoring' (to deal with equivalence) similar to the algorithm proposed by Ng [28],
- BwFw defines potential domains of predicates (as in Magic Sets [33]), guesses all potential atoms as abducibles, infers truth of atoms using the original axioms from the knowledge base, and checks if this explains the observation,
- Simpl realizes a simplified abduction with closed domain and UNA.

BackCh and BwFw realizing existential quantification using uninterpreted functions to build Skolem terms. Simpl serves as a performance baseline.

Our experiments with Accel show, that only small instances can be solved within reasonable resource limits, and that memory as well as proving optimality of solutions are problematic issues. We analyze the main reasons for these observations using solver statistics, and we outline possibilities for achieving better results in future work.

Section 2 gives preliminaries of abduction and ASP, Section 3 describes rewritings and ASP encodings, Section 4 reports on experiments, and Section 5 concludes with related work and an outlook on future work.

2 Preliminaries

We give a brief introduction of Abduction in general and First Order Horn abduction in specific, moreover we give brief preliminaries of ASP.

Notation: variables start with capital letters and constants with small letters.

2.1 Abduction

Abduction, originally described in [29], can be defined logically as follows. Given a set T of background knowledge axioms and an observation O, find a set H

[1] This should be true according to [28]. Actually we had to deactivate one cyclic axiom in the knowledge base to make this property true.

of hypothesis atoms such that T and H are consistent, and reproduce the observation, i.e., $T \cup H \nvDash \bot$ and $T \cup H \vDash O$. In this work we formalize axioms and observations in First Order Logic (FOL) as done by Ng [28]: the observation (in the following called 'goal') O is an existentially quantified conjunction of atoms

$$\exists V_1, \dots, V_k : o_1(V_1, \dots, V_k) \wedge \dots \wedge o_m(V_1, \dots, V_k) \tag{1}$$

and an axiom in T is a universally quantified Horn clause of form

$$c(V_1, \dots, V_k) \Leftarrow p_1(V_1, \dots, V_k) \wedge \dots \wedge p_r(V_1, \dots, V_k). \tag{2}$$

or an integrity constraints (like (2) but without a head).

The set H of hypotheses can contain any predicate from the theory T and the goal O, hence existence of a solution is trivial. Explanations with minimum cardinality are considered optimal. A subset of constants is declared as sort names that cannot be abduced as equivalent with other constants.

Example 1 (Running Example). Consider the following text

'*Mary lost her father. She is depressed.*'

which can be encoded as the following FOL goal, to be explained by abduction.

$$\mathit{name}(m, \mathit{mary}) \wedge \mathit{lost}(m, f) \wedge \mathit{fatherof}(f, m) \wedge \mathit{inst}(s, \mathit{female}) \wedge \mathit{is}(s, \mathit{depressed})$$

Given the set of axioms

$$\begin{aligned}
\mathit{inst}(X, \mathit{male}) &\Leftarrow \mathit{fatherof}(X, Y) && (3)\\
\mathit{inst}(X, \mathit{female}) &\Leftarrow \mathit{name}(X, \mathit{mary}) && (4)\\
\mathit{importantfor}(Y, X) &\Leftarrow \mathit{fatherof}(Y, X) && (5)\\
\mathit{inst}(X, \mathit{person}) &\Leftarrow \mathit{inst}(X, \mathit{male}) && (6)\\
\mathit{is}(X, \mathit{depressed}) &\Leftarrow \mathit{inst}(X, \mathit{pessimist}) && (7)\\
\mathit{is}(X, \mathit{depressed}) &\Leftarrow \mathit{is}(Y, \mathit{dead}) \wedge \mathit{importantfor}(Y, X) && (8)\\
\mathit{lost}(X, Y) &\Leftarrow \mathit{is}(Y, \mathit{dead}) \wedge \mathit{importantfor}(Y, X) \wedge \mathit{inst}(Y, \mathit{person}) && (9)
\end{aligned}$$

and sort names

$$\mathit{person} \quad \mathit{male} \quad \mathit{female} \quad \mathit{dead} \quad \mathit{depressed} \tag{10}$$

we can use abduction to infer the following: (a) loss of a person here should be interpreted as death, (b) 'she' refers to Mary, and (c) her depression is because of her father's death because her father was important for her.

This is reached by abducing the following atoms and equivalences.

$$\mathit{name}(m, \mathit{mary}) \quad \mathit{fatherof}(f, m) \quad \mathit{is}(f, \mathit{dead}) \quad m = s \tag{11}$$

The first two atoms directly explain goal atoms. We can explain the remaining goal atoms by showing inferring truth of the following atoms.

$$inst(f, male) \quad \text{[infered via (3) using (11)]} \tag{12}$$

$$inst(m, female) \quad \text{[infered via (4) using (11)]} \tag{13}$$

$$inst(s, female) \quad \text{[goal, factored from (13) using (11)]}$$

$$importantfor(f, m) \quad \text{[infered via (5) using (11)]} \tag{14}$$

$$inst(f, person) \quad \text{[infered via (6) using (12)]} \tag{15}$$

$$is(m, depressed) \quad \text{[infered via (8) using (11) and (14)]} \tag{16}$$

$$is(s, depressed) \quad \text{[goal, factored from (16) using (11)]}$$

$$lost(m, f) \quad \text{[goal, infered via (9) using (11), (14), and (15)]}$$

Note that there are additional possible inferences but they are not necessary to explain the goal atoms. Moreover note that another abductive explanations (with higher number of abduced atoms and therefore higher cost) would be to abduce all goal atoms, or to abduce $inst(m, pessimist)$ and $lost(m, f)$ instead of abducing $is(f, dead)$. □

Complexity-wise we think that deciding optimality of a solution is harder than for cardinality minimal abduction on (non-ground) logic programs under well-founded semantics [11, Sec. 4.1.3] because value invention provides a ground term inventory of polynomial size in the number of constants in the goal. The propositional case has been analyzed in [3, 10]. See also Section 5.1.

2.2 Answer Set Programming

We assume familiarity with ASP [16, 25, 12, 14] and give only brief preliminaries of those part of the ASP-Core-2 standard [5] that we will use: logic programs with (uninterpreted) function symbols, aggregates, choices, and weak constraints. A logic program consists of rules of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_n, \mathbf{not}\ \beta_{n+1}, \ldots, \mathbf{not}\ \beta_m.$$

where α and β_i are head and body atoms, respectively, and **not** denotes negation as failure. We say that a rule is a fact if $m = 0$, and it is a constraint if there is no head α. Atoms can contain constants, variables, and function terms, and programs must obey safety restrictions (see [5]) to ensure a finite instantiation. Anonymous variables of form '_' are replaced by new variable symbols.

An aggregate literal in the body of a rule accumulates truth values from a set of atoms, e.g., $2 = \#count\{X : p(X)\}$ is true iff the extension of p (in the answer set candidate) contains exactly 2 elements.

Choice constructions can occur instead of rule heads, they generate a solution space if the rule body is satisfied; e.g., $1 \leq \{p(a); p(b); p(c)\} \leq 2$ in the rule head generates all solution candidates where at least 1 and at most 2 atoms of the set are true. The bounds can be omitted. The colon symbol ':' can be used to define conditions for including atoms in a choice, for example the choice

$\{p(X) : q(X), \textbf{not } r(X)\}$ encodes a guess over all $p(X)$ such that $q(X)$ is true and $r(X)$ is not true.

A weak constraint of form

$$\Leftarrow p(X). \quad [1@1, X]$$

denotes that an answer set I has cost equivalent to the size of the extension of p in I. Answer sets of the lowest cost are considered optimal. Note that the syntax $[A@B, X]$ denotes that cost A is incurred on level B, and costs are summed over all distinct X (i.e., X ensures each atom $p(X) \in I$ is counted once).

Semantics of an answer set program P are defined using its Herbrand universe, ground instantiation, and the GL-reduct [16] which intuitively reduces the program using assumptions in an answer set candidate.

3 Modeling Abduction in ASP

We next describe two encodings for modeling abduction with partial UNA and with value invention in ASP (Sections 3.1 and 3.2) and one simpler encoding with UNA for all constants and without value invention (Section 3.3).

All atoms in Accel have arity 2. For practical reasons we represent an atom of the form $pred(arg1, arg2)$ as $c(pred, arg1, arg2)$.

Goals in Accel have no variables, hence we can represent them as facts. For our example we represent the goal as the following set of ASP facts.

$$goal(c(name, m, mary)). \quad goal(c(lost, m, f)). \quad goal(c(fatherof, f, m)).$$
$$goal(c(inst, s, female)). \quad goal(c(is, s, depressed)).$$

Sorts are encoded as facts as follows.

$$sortname(person). \quad sortname(male). \quad sortname(female).$$
$$sortname(dead). \quad sortname(depressed).$$

3.1 Modeling Backward-Chaining (BackCh)

Our first encoding represents the abduction algorithm by Ng [28] in the ASP grounder: (i) backward-chain from the goal and invent new constants on the way, (ii) factor atoms with other atoms in the goal or with atoms discovered in (i) and abduce equivalences between constant terms, (iii) mark atoms that have not been inferred or factored as abduced, and (iv) search for the solution with the minimum number of abduced atoms while obeying integrity constraints. Note that this is the most straightforward modeling idea, although we consider it the least declarative one because it is oriented towards realizing an algorithm and does not realize abduction as usually done in ASP (for that see next section).

First, everything that is a goal is defined to be true, and everything true must either be abduced, inferred, or factored.

$$\begin{aligned} true(P) &\leftarrow goal(P). \\ 1 \leq \{abduce(P); infer(P); factor(P)\} \leq 1 &\leftarrow true(P). \end{aligned} \tag{17}$$

We prefer solutions with a minimum number of abduced terms.

$$\leftsquigarrow \mathit{abduce}(P). \quad [1@1, P] \tag{18}$$

Backward chaining is realized by rewriting each axiom of form (2) into two parts: a guess whether the head is inferred via this rule, and for each body atom that it must be inferred, abduced, or factored, if the head was inferred.

For example axiom (8) is translated into

$$\begin{aligned}
&0 \leq \{\mathit{inferVia}(r_1, c(\mathit{is}, X, \mathit{depressed}))\} \leq 1 \leftarrow \\
&\qquad\qquad\qquad \mathit{infer}(c(\mathit{is}, X, \mathit{depressed})). &(19)\\
&\mathit{inferenceNeeds}(c(\mathit{is}, X, \mathit{depressed}), r_1, c(\mathit{importantfor}, s_1(X), X)) \leftarrow \\
&\qquad\qquad\qquad \mathit{inferVia}(r_1, c(\mathit{is}, X, \mathit{depressed})). &(20)\\
&\mathit{inferenceNeeds}(c(\mathit{is}, X, \mathit{depressed}), r_1, c(\mathit{is}, s_1(X), \mathit{dead})) \leftarrow \\
&\qquad\qquad\qquad \mathit{inferVia}(r_1, c(\mathit{is}, X, \mathit{depressed})). &(21)
\end{aligned}$$

where r_1 is a unique identifier for axiom (8); rule (19) guesses if inferring $\mathit{is}(X, \mathit{depressed})$ happens via r_1; and rules (20) and (21) define that this inference requires to justify atoms $\mathit{importantfor}(s_1(X), X)$ and $\mathit{is}(s_1(X), \mathit{dead})$. Note that $s_1(\cdot)$ is a Skolem function unique to axiom (8).

For each atom that is inferred, we add a constraint that it must be inferred via at least one rule. Moreover each atom required by such inference is defined as true and hence must be justified according to (17).

$$\begin{aligned}
&\leftarrow infer(P),\ 0 \leq \#count\{A, P : inferVia(A, P)\} \leq 0.\\
true(Body) &\leftarrow inferenceNeeds(Head, Axiom, Body).
\end{aligned}$$

For factoring we need to handle absence of the UNA, so we obtain the Herbrand Universe (HU) and obtain the 'User Herbrand Universe' (UHU) of constants that are not sort names and can be equivalent to other constants.

$$\begin{aligned}
hu(C) &\leftarrow true(c(\_, C, \_)).\\
hu(C) &\leftarrow true(c(\_, \_, C)).\\
uhu(C) &\leftarrow hu(C), \textbf{not}\ sortname(C).
\end{aligned}$$

We guess if a member of UHU is represented by another member of UHU.

$$0 \leq \{rep(X, Y) : uhu(X), X < Y\} \leq 1 \leftarrow uhu(Y). \tag{22}$$

This guesses at most one representative for each member of UHU. Note that operator '<' in (22) realizes lexicographic comparison of ground terms; this means the representative of an equivalence class of UHU terms is always the smallest term in that class (this reduces symmetries in the search space).

The following rules ensure that no ground term is both represented and representing another one using *rep*.

$$\begin{aligned}
representative(X) &\leftarrow rep(X, Y). &(23)\\
represented(Y) &\leftarrow rep(X, Y). &(24)\\
&\leftarrow representative(X), represented(X). &(25)
\end{aligned}$$

Rules (22)–(25) generate all possible equivalence relations over UHU terms, encoded as mappings to representative terms. If a term has neither a representative nor is representing another term, it is a singleton equivalence class.

Given rep, we define a mapping map for all HU terms to their representative, where singletons and sort names are their own representatives.

$$map(X,Y) \leftarrow rep(X,Y). \tag{26}$$

$$map(X,X) \leftarrow hu(X), \mathbf{not}\ represented(X). \tag{27}$$

Now that we generate and represent equivalence classes, we can perform factoring. We experiment with three formulations: all of them define a predicate $factorOk(P)$ if factoring is allowed, and they share the following three rules: (28) requires $factorOk$ to be true for factored atoms, while (29) and (30) define the *factoring base* which is the set of inferred or abduced atoms, i.e., the set of atoms that other atoms can be factored with.

$$\leftarrow factor(P),\ \mathbf{not}\ factorOk(P). \tag{28}$$

$$factoringbase(A) \leftarrow infer(A). \tag{29}$$

$$factoringbase(A) \leftarrow abduce(A). \tag{30}$$

The three factoring variations are as follows.

• Factoring (a) uses the map predicate to match factored atoms to the factoring base and is realized as follows.

$$\begin{aligned} factorOk(c(P,A_1,B_1)) \leftarrow\ & factor(c(P,A_1,B_1)), factoringbase(c(P,A_2,B_2)),\\ & map(A,A_1), map(A,A_2), map(B,B_1), map(B,B_2). \end{aligned}$$

• Factoring (b) defines a canonical factoring base using the map predicate and matches factored elements with that canonical base using the following rules.

$$\begin{aligned} cfactoringbase(c(P,A,B)) &\leftarrow factoringbase(c(P,A_1,B_1)),\\ &\quad\ map(A,A_1), map(B,B_1).\\ factorOk(c(P,A_1,B_1)) &\leftarrow cfactoringbase(c(P,A,B)),\\ &\quad\ factor(c(P,A_1,B_1)), map(A,A_1), map(B,B_1). \end{aligned}$$

• Factoring (c) defines a relation for canonicalizing atoms using the map predicate and defines $factorOk$ using that relation.

$$\begin{aligned} noncanonical(P) &\leftarrow factor(P).\\ noncanonical(Q) &\leftarrow factoringbase(Q).\\ canonical(c(P,A,B), c(P,A_1,B_1)) &\leftarrow noncanonical(c(P,A_1,B_1)),\\ &\quad\ map(A,A_1), map(B,B_1).\\ factorOk(P_1) &\leftarrow factor(P_1), factoringbase(P_2),\\ &\quad\ canonical(P,P_1), canonical(P,P_2). \end{aligned}$$

3.2 Skolemized Domain and Standard ASP Abduction (BwFw)

This encoding follows an idea similar to Magic Sets [33]: starting from the goal (in Magic Sets the query) we represent the domain of each argument of each predicate. Subsequently we define domains of predicates in the body of an axiom based on the domain of the predicate in heads of that axiom, deterministically expanding the domain with Skolem terms whenever there are variables in the axiom body that are not in the axiom head. Once we have the domains, we can follow the usual ASP approach for abduction: (i) guess (using the domain) which atoms are our abduction hypothesis, (ii) use axioms to infer what becomes true due to the abduction hypothesis; and (iii) require that the observation is reproduced. In BackCh, equivalence was used for factoring (which reduces the number of abducibles required to derive the goal). In BwFw, we instantiate the whole potential proof tree, so we do not need factoring between atoms in rules, it is sufficient to factor abducibles with one another. Hence in this encoding we handle equivalences by defining an atom to be true if its terms are equivalent with the terms of an abduced atom.

We next give the encoding in detail.

Predicates in Accel have arity 2, so we represent the domain as $dom(P,S,O)$, i.e., predicate P has $\langle S,O\rangle$ as potential extension. We seed dom from the goal.

$$dom(P,S,O) \leftarrow goal(c(P,S,O)).$$

Domains are propagated by rewriting axioms of form (2) as indicated above. For example (8) is translated into the following rules.

$$dom(importantfor, s_1(X), X) \leftarrow dom(is, X, depressed). \tag{31}$$

$$dom(is, s_1(X), dead) \leftarrow dom(is, X, depressed). \tag{32}$$

Additionally we rewrite each axiom into an equivalent rule in the ASP representation, for example we rewrite (8) into the following rule.

$$\begin{aligned} true(c(is,X,depressed)) \leftarrow\; & true(c(importantfor,Y,X)), \\ & true(c(is,Y,dead)). \end{aligned} \tag{33}$$

For guessing representatives we first define UHU for first and second arguments of each predicate and we define HU over all arguments.

$$\begin{aligned} dom_1(P,X) &\leftarrow dom(P,X,\_). \\ dom_2(P,Y) &\leftarrow dom(P,\_,Y). \\ uhu_1(P,X) &\leftarrow dom_1(P,X), \mathbf{not}\ sortname(X). \\ uhu_2(P,Y) &\leftarrow dom_2(P,Y), \mathbf{not}\ sortname(Y). \\ hu(X) &\leftarrow dom_1(\_,X). \\ hu(Y) &\leftarrow dom_2(\_,Y). \end{aligned}$$

We guess representatives, i.e., equivalence classes, only among those UHU elements that can potentially be unified because they are arguments of the same predicate at the same argument position.[2]

$$0 \leq \{rep(X_1, X_2) : uhu_1(P, X_2), X_1 < X_2\} \leq 1 \leftarrow uhu_1(P, X_1).$$
$$0 \leq \{rep(Y_1, Y_2) : uhu_2(P, Y_2), Y_1 < Y_2\} \leq 1 \leftarrow uhu_2(P, Y_1).$$

We reuse (23)–(27) from the BackCh encoding to ensure that *rep* encodes an equivalence relation and to define *map*. (Note also that *hu* is used in (27).)

We abduce atoms using the domain under the condition that the domain elements are representatives of their equivalence class (symmetry breaking).

$$\{abduce(c(P,S,O)) : dom(P,S,O), \textbf{not } represented(S), \textbf{not } represented(O)\}.$$

We define that abduced atoms are true, and we use *map* to define that atoms equivalent with abduced atoms are true.

$$true(A) \leftarrow abduce(A).$$
$$true(c(P,A,B)) \leftarrow abduce(c(P,RA,RB)), map(RA,A), map(RB,B). \quad (34)$$

This makes all consequences of the abduction hypothesis in the axiom theory true while taking into account equivalences.

Finally we again require that the observation is reproduced, and we again minimize the number of abduced atoms. (Note that term equivalence has been taken care of in (34) hence we can ignore it for checking the goal.)

$$\leftarrow goal(A), \textbf{not } true(A). \quad (35)$$
$$\leftharpoondown abduce(P). \quad [1@1, P] \quad (36)$$

3.3 Closed Domain and UNA (Simpl)

This encoding propagates domains differently from the previous one: it does not introduce Skolem terms but always the same *null* term for variables in axiom bodies that are not contained in the axiom head. Abduction substitutes all possible constants for these *null* terms, hence this encoding does not realize an open domain. Moreover, this encoding uses the UNA for all terms.

This encoding is less expressive than the previous ones, it is incomplete but sound: Simpl finds a subset of solutions that other encodings find, and for this subset each solution has same cost as in other encodings. We use Simpl primarily for comparing memory and time usage with other encodings and other approaches. For readability we repeat some rules from previous encodings.

As in BwFw, the domain is seeded from the goal.

$$dom(P,S,O) \leftarrow goal(c(P,S,O)).$$

The rewriting is different, however: instead of rewriting the example axiom (8) into (31) and (32) we create the following rules.

$$dom(importantfor, null, X) \leftarrow dom(is, X, depressed).$$
$$dom(is, null, dead) \leftarrow dom(is, X, depressed).$$

[2] A more naive encoding showed significantly higher memory requirements.

	Status				Resources	
Encoding	OPT sum	SAT sum	OOT sum	OOM sum	Space (osp) MB/avg	Time (otm/grd/slv) s/avg
BackCh (a)	40	0	0	460	4636 (1375)	148 (152/142/ 6)
BackCh (b)	**238**	2	0	260	3537 (2339)	175 (147/135/ 41)
BackCh (c)	36	4	0	460	4612 (1146)	126 (152/117/ 9)
BwFw	10	0	0	490	4884 (524)	212 (12/211/ 0)
Simpl	132	258	10	100	1875 (208)	380 (107/ 52/323)

Table 1. Experimental Results, accumulated over 10 runs of each of the 50 instances of Accel: OPT, SAT, OOT, and OOM denote that the solver found the optimal solution, found a solution without proving optimality, exceeded the time limit (10 min), and exceeded the memory (5 GB), respectively. Numbers in brackets are the space and time usage of runs that found the optimum (osp/otm), the time spent in grounding (grd), and the time spent in solving (slv). Averages are computed over all runs, except for (osp) and (otm) that are averaged over runs where the optimum was found.

Here, *null* is used instead of Skolem terms.

We define the HU as all domain elements except *null*.

$$\begin{aligned} hu(X) &\leftarrow dom(\_, X, \_), X \neq null. \\ hu(Y) &\leftarrow dom(\_, \_, Y), Y \neq null. \end{aligned}$$

Now we define a mapping that maps *null* to each element of (implicit) UHU and contains the identity mapping for HU.

$$\begin{aligned} nullrepl(X, X) &\leftarrow hu(X). \\ nullrepl(null, X) &\leftarrow hu(X), \mathbf{not}\ sortname(X). \end{aligned} \tag{37}$$

We abduce atoms using this mapping and define that abduced atoms are true.

$$\begin{aligned} \{abduce(c(P, S', O'))\} &\leftarrow dom(P, S, O), nullrepl(S, S'), nullrepl(O, O'). \\ true(c(P, A, B)) &\leftarrow abduce(c(P, A, B)). \end{aligned}$$

We propagate truth as in BwFw using the rewriting exemplified in (33).

We again require goals to be true and minimize the number of abduced atoms.

$$\begin{aligned} &\leftarrow goal(A),\ \mathbf{not}\ true(A). \\ &\leftsquigarrow abduce(P). \quad [1@1, P] \end{aligned}$$

Note that, although this encoding solves an easier problem than the other encodings, we will see that it does not necessarily perform better.

4 Experimental Results

To test the above encodings, we performed experiments using the Accel benchmark[3] [28]. This benchmark consists of 50 instances (i.e., goals) with between 5

[3] ftp://ftp.cs.utexas.edu/pub/mooney/accel

and 26 atoms in a goal (12.6 atoms on average), and a knowledge base consisting of 190 first order Horn rules with bodies of size between 1 and 11 atoms (2.2 on average). Our encodings and instances used in experiments are available online.[4]

The benchmarks were performed on a computer with 48 GB RAM and two Intel E5-2630 CPUs (total 16 cores) using Ubuntu 14.04 and Clingo 4.5.0 [15]. Each run was limited to 10 minutes and 5 GB RAM, HTCondor was used as a job scheduling system, each run was repeated 10 times and no more than 8 jobs were running simultaneously. For Clingo we used the setting `--configuration=handy`.

Table 1 shows results of the experiments.

Exceeding the memory of 5 GB turns out to be a significant problem in all encodings, even for Simpl where (only) 10 instances exceeding the memory. Memory was always exceeded during grounding, never during solving.

Our encodings perform much worse than the state-of-the-art for solving Accel which is less than 10 s per instance on average [19].

Encoding Simpl is able to ground more instances than the other encodings, however it only finds the optimal solution for 132 runs while BackCh (b) finds optimal solutions for 238 runs, although Simpl encodes a sound approximation of the problem encoded in BackCh and BwFw.

Encoding BwFw, which is most similar to classical abduction encodings in ASP, performs worst due to memory exhaustion: only one instance can be solved. In BwFw the high space complexity comes from the full forward-instantiation of the whole knowledge base after guessing representatives for the domain of each abduced atom. Although BwFw first backward-chains the domain using a deterministic set of rules, it seems to instantiate a much larger part of the rules necessary for solving the problem, in particular in comparison with the BackCh encoding. Observe that we split the representation for UHU which might seem unintuitive as we do not do this in other encodings. However, when we experimented with a more naive encoding for BwFw (replacing uhu_1 and uhu_2 by a single uhu) this encoding could not even solve a single instance within the 5 GB memory limit.

The BackCh encodings perform best and show big differences among the factoring variations. Factoring (b) is clearly superior to (a) and (c): it uses significantly less memory (only 260 out-of-memory runs) and once grounding is successful it solves many instances (238) within the timeout. The reason for this can be found in the rules defining *factorOk*: factoring (a) contains 4 atoms of *map* with six joining 6 variables, hence its instantiation contains (in worst case) $\mathcal{O}(n^6)$ rules, where n is the size of HU; moreover (c) defines *canonical* from *map* with by defining $\mathcal{O}(n^3)$ distinct atoms and joins these atoms to define *factorOk*, again resulting in $\mathcal{O}(n^6)$ rules. Encoding (b) on the other hand defines $\mathcal{O}(n^3)$ distinct atoms with predicate *cfactoringbase* and joins them to 2 instances of *map*, resulting in $\mathcal{O}(n^5)$ rules defining *factorOk*.

Single Instance Analysis. To analyze performance differences between encodings, we looked at instance # 8 which is the smallest instance and the single instance where all encodings found an optimal solution.

[4] `https://bitbucket.org/knowlp/supplement-rcra2015-asp-fo-abduction/`

Encodings BackCh (a) and BackCh (c) require 117 s and 244 s grounding time, respectively, and both groundings contain ~150 k ASP rules. Although BackCh (b), BwFw, and Simpl stay below 10 s grounding time, BwFw produces an amazing 803 k ASP rules. The reason is, that (34) creates many potentially true atoms, which instantiates many axioms. As a consequence BwFw grounds comparatively fast but at the same time produces the biggest grounding.

The smallest grounding is produced by Simpl, which also has fewest OOM results. Yet Simpl finds the optimal solution for fewer instances than BackCh (b). Solver statistics show that Simpl requires 463 k choice points in the solver and creates 4 k conflict lemmas for instance 8, while BackCh (b) proves optimality using 737 choice points and 150 lemmas. We conjecture that these choice points originate in the naive substitution of *null* with UHU elements in (37), which produces a big grounding and a big hypothesis space with many symmetric solutions of equal cost. Such symmetries makes it difficult to prove optimality.

Additional Observations. As suggested by a reviewer we ran our encodings with and without *projection* to *abduce* atoms. This did not change run times significantly, however it greatly reduced log file size.

5 Discussion and Conclusion

We presented encodings for realizing abduction with an open domain in the absence of the UNA and performed experiments using the Accel benchmark.

Experiments show clear differences between our encodings, and they all perform much worse than the state-of-the-art for Accel [19]. Hence we consider this work a negative result. Nevertheless we observed huge performance variation between our encodings, and we think there is still potential for finding better encodings (and solver parameters).

5.1 Related Work

The idea of abduction goes back to Peirce [29] and was later formalized in logic.

Abductive Logic Programming (ALP) is an extension of logic programs with abduction and integrity constraints. Kakas et al. [20] discuss ALP and applications, in particular they relate Answer Set Programming and abduction. Fung et al. describe the IFF proof procedure [13] which is a FOL rewriting that is sound and complete for performing abduction in a fragment of ALP with only classical negation and specific safety constraints. Denecker et al. [8] describe SLDNFA-resolution which is an extension of SLDNF resolution for performing abduction in ALP in the presence of negation as failure. They also describe a way to 'avoid Skolemization by variable renaming' which is a strategy that can be mimicked in ASP by using HEX for value invention (instead of uninterpreted function terms). Kakas et al. describe the $\mathcal{A}$-System for evaluating ALP using an algorithm that interleaves instantiation of variables and constraint solving [21]. The CIFF framework [26] is conceptually similar to the $\mathcal{A}$-System but it allows a more relaxed use of negation. The $\mathcal{S}$CIFF framework [1] relaxes some restrictions of CIFF and provides facilities for modeling agent interactions.

Implementations of ALP have in common that they are based on evaluation strategies similar to Prolog [26]. CIFF is compare with ASP on the example of n-queens and the authors emphasize that CIFF has more power due to its partial non-ground evaluation [26]. However they also show a different CIFF encoding that performs much worse than ASP. Different from CIFF and earlier ALP implementations, our ASP encodings are first instantiated and then solved.

Weighted abduction (called cost-based abduction by some authors) [18] is FOL abduction with costs that are propagated along axioms in a proof tree. The purpose of costs is to find the most suitable abductive explanations for interpretation of natural language. The solver Henry-n700 [19] realizes weighted abduction by first instantiating an ILP instance with Java and then finding optimal solutions for the ILP instance. Our approach is similar and therefore our work is related more to weighted abduction than to ALP.

Probabilistic abduction was realized in Markov Logic [30] in the Alchemy system [23] although without an open domain [2, 32], which makes it similar to the Simpl encoding. (Alchemy naively instantiates existential variables in rule heads with all known ground terms just as Simpl does.)

5.2 Future Work

As the Accel benchmark problems can be solved much faster (i.e., within a few seconds on average) with a classical backward chaining implementation in Java and an ILP solver [19], we conjecture that a solution in ASP has the potential to be similarly fast. In the future we want to develop encodings that perform nearer to the state-of-the-art. We believe that we can gain increased flexibility if we model this problem in ASP as opposed to creating an ILP theory using Java.

Our experiments show that small changes in encodings can lead to big performance changes regarding grounding size, grounding speed, finding an initial solution, and proving optimality. We are currently investigating alternative encodings, in particular to replace guessing of representatives by a more direct guessing of the equivalence relation.

As large groundings are a major issue in our work, interesting future work includes using dlv [24] which is known to be strong in grounding, or solvers that perform lazy grounding such as IDP [7] or OMiGA [6]. Another possible improvement of grounding is to replace grounding-intensive constraints by sound approximations with lower space complexity, and create missing constraints on demand when they are violated. An equivalent strategy is used in Henry-n700 where and it is called Cutting Plane Inference and described as essential for the performance of the ILP solution for weighted abduction [19]. In ASP such strategies are used by state-of-the-art solvers for on-demand generation of loop-formula nogoods. User-defined on-demand constraints can be realized using a custom propagator in Clingo [15] or on-demand constraints in HEX [9, Sec. 2.3.1].

Potentially beneficial for future improvements of this work are Open Answer Set Programming (OASP) [17] which inherently supports open domains, and

Datalog$^{\pm}$ [4] which supports existential constructions in rule heads. Both formalisms have been used for representing Description Logics in ASP and could provide insights for future work about modeling FOL abduction in ASP.

So far we prefer the least number of abduced atoms to find optimal solutions. In the future we want to consider more sophisticated preferences based on Coherence [27] and based on Relevance Theory [31].

Acknowledgements

We thank the anonymous reviewers for their constructive and useful feedback.

References

1. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic*, 9(4):Article No. 29, 2008.
2. J. Blythe, J. R. Hobbs, P. Domingos, R. J. Kate, and R. J. Mooney. Implementing Weighted Abduction in Markov Logic. In *International Conference on Computational Semantics (IWCS)*, pages 55–64, 2011.
3. T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49(1-3):25–60, 1991.
4. A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog+/-: A Unified Approach to Ontologies and Integrity Constraints. In *International Conference on Database Theory*, pages 14–30, 2009.
5. F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-Core-2 Input language format. Technical report, ASP Standardization Working Group, 2012.
6. M. Dao-tran, T. Eiter, M. Fink, G. Weidinger, and A. Weinzierl. OMiGA : An Open Minded Grounding On-The-Fly Answer Set Solver. In *Logics in Artificial Intelligence (JELIA)*, pages 480–483, 2012.
7. B. De Cat, M. Denecker, P. Stuckey, and M. Bruynooghe. Lazy model expansion: Interleaving grounding with search. *Journal of Artificial Intelligence Research*, 52:235–286, 2015.
8. M. Denecker and D. de Schreye. SLDNFA: An abductive procedure for abductive logic programs. *The Journal of Logic Programming*, 34(2):111–167, 1998.
9. T. Eiter, M. Fink, G. Ianni, T. Krennwallner, C. Redl, and P. Schüller. A model building framework for answer set programming with external computations. *Theory and Practice of Logic Programming*, 2015. arXiv:1507.01451 [cs.AI], to appear.
10. T. Eiter and G. Gottlob. The Complexity of Logic-Based Abduction. *Journal of the ACM*, 42(1):3–42, 1995.
11. T. Eiter, G. Gottlob, and N. Leone. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, 189(1-2):129–177, 1997.
12. T. Eiter, G. Ianni, and T. Krennwallner. Answer Set Programming: A Primer. In *Reasoning Web Summer School*, Lecture Notes in Computer Science, 2009.
13. T. H. Fung and R. Kowalski. The IFF proof procedure for abductive logic programming. *The Journal of Logic Programming*, 33(2):151–165, 1997.
14. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Morgan Claypool, 2012.

15. M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, 24(2):107–124, 2011.
16. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *International Conference and Symposium on Logic Programming (ICLP/SLP)*, pages 1070–1080, 1988.
17. S. Heymans, D. Van Nieuwenborgh, and D. Vermeir. Open Answer Set Programming with Guarded Programs. *ACM Transactions on Computational Logic*, 9(4):26, 2008.
18. J. R. Hobbs, M. Stickel, P. Martin, and D. Edwards. Interpretation as abduction. *Artificial Intelligence*, 63(1-2):69–142, 1993.
19. N. Inoue and K. Inui. ILP-based Inference for Cost-based Abduction on First-order Predicate Logic. *Information and Media Technologies*, 9:83–110, 2014.
20. A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
21. A. C. Kakas, B. Van Nuffelen, and M. Denecker. A-System: Problem solving through abduction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 591–596, 2001.
22. R. J. Kate and R. J. Mooney. Probabilistic Abduction using Markov Logic Networks. In *IJCAI Workshop on Plan, Activity, and Intent Recognition*, 2009.
23. S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, L. D, J. Wang, A. Nath, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, 2010.
24. N. Leone, G. Pfeifer, W. Faber, and T. Eiter. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):1–58, 2006.
25. V. Lifschitz. What Is Answer Set Programming ? In *AAAI Conference on Artificial Intelligence*, pages 1594–1597, 2008.
26. P. Mancarella, G. Terreni, F. Sadri, F. Toni, and U. Endriss. The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments. *Theory and Practice of Logic Programming*, 9(6):691–750, 2009.
27. H. Ng and R. Mooney. Abductive Plan Recognition and Diagnosis: A Comprehensive Empirical Evaluation. In *Knowledge Representation and Reasoning (KR)*, pages 499–508, 1992.
28. H. T. Ng. *A General Abductive System with Applications to Plan Recognition and Diagnosis.* Phd thesis, University of Texas at Austin, 1992.
29. C. S. Peirce. Abduction and Induction. In *Philosophical Writings of Peirce*, chapter 11, pages 150–156. Dover Publications, 1955.
30. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, Jan. 2006.
31. P. Schüller. Tackling Winograd Schemas by Formalizing Relevance Theory in Knowledge Graphs. In *International Conference on Principles of Knowledge Representation and Reasoning (KR).* AAAI Press, 2014.
32. P. Singla and R. J. Mooney. Abductive Markov Logic for Plan Recognition. In *AAAI Conference on Artificial Intelligence*, pages 1069–1075, 2011.
33. J. D. Ullman. *Principles of Database and Knowledge Base Systems.* Computer Science Press, 1989.