TLP **17** (4): 559–590, 2017. © Cambridge University Press 2017
doi:10.1017/S1471068417000114 First published online 28 June 2017

*On relation between constraint answer set programming and satisfiability modulo theories**

YULIYA LIERLER and BENJAMIN SUSMAN
Department of Computer Science, University of Nebraska at Omaha
Omaha, NE 68182, USA
(*e-mails:* ylierler@unomaha.edu, bsusman@unomaha.edu)

submitted 16 April 2016; revised 28 February 2017; accepted 25 March 2017

Abstract

Constraint answer set programming is a promising research direction that integrates answer set programming with constraint processing. It is often informally related to the field of satisfiability modulo theories. Yet, the exact formal link is obscured as the terminology and concepts used in these two research areas differ. In this paper, we connect these two research areas by uncovering the precise formal relation between them. We believe that this work will boost the cross-fertilization of the theoretical foundations and the existing solving methods in both areas. As a step in this direction, we provide a translation from constraint answer set programs with integer linear constraints to satisfiability modulo linear integer arithmetic that paves the way to utilizing modern satisfiability modulo theories solvers for computing answer sets of constraint answer set programs.

KEYWORDS: constraint answer set programming, constraint satisfaction processing, satisfiability modulo theories

1 Introduction

Constraint answer set programming (CASP) (Elkabani *et al.* 2004; Mellarkod *et al.* 2008; Balduccini 2009; Gebser *et al.* 2009; Lierler 2014) is a promising research direction that integrates answer set programming, a powerful knowledge representation paradigm, with constraint processing. Typical answer set programming tools start their computation with grounding, a process that substitutes variables for passing constants in respective domains. Large domains often form an obstacle for classical answer set programming. CASP enables a mechanism to model constraints over large domains so that they are processed in a non-typical way for answer set programming tools by delegating their solving to constraint solver systems specifically designed to handle large and sometimes infinite domains. CASP solvers including CLINGCON (Gebser *et al.* 2009) and EZCSP (Balduccini 2009) already put CASP on the map of efficient automated reasoning tools.

* This is an extended version of the paper that appeared at IJCAI-2016 (Lierler and Susman 2016).

CASP often cites itself as a related initiative to satisfiability modulo theories (SMT) solving (Barrett and Tinelli 2014). Yet, the exact link is obscured as the terminology and concepts used in both fields differ. To add to the complexity of the picture, several answer set programming modulo theories formalisms have been proposed. For instance, Liu *et al.* (2012), Janhunen *et al.* (2011), and Lee and Meng (2013) introduced logic programs modulo linear constraints, logic programs modulo difference constraints, and ASPMT programs, respectively. Acyclicity programs (Bomanson *et al.* 2015) (or logic programs modulo acyclicity constraints) form another recently investigated formalism that parallels satisfiability modulo graphs framework developed within SMT (Gebser *et al.* 2014).

This work attempts to unify the terminology used in CASP and SMT so that the differences and similarities of logic programs with constraints versus logic programs modulo theories become apparent. At the same time, we introduce the notion of constraint formulas, which is similar to that of logic programs with constraints. We identify a special class of SMT theories that we call "uniform." Commonly used theories in satisfiability modulo solving such as integer linear, difference logic, and linear arithmetics belong to uniform theories. This class of theories helps us to establish precise links (i) between CASP and SMT, and (ii) between constraint formulas and SMT formulas. We are able to then provide a formal description relating a family of distinct CASP formalisms.

We show that this unified outlook allows us not only to better understand the landscape of CASP languages and systems, but also to foster new ideas for the design of CASP solvers and possibly SMT solvers. For example, theoretical results of this work establish a simple method for using SMT systems for computing answer sets of a broad class of tight constraint answer set programs. Susman and Lierler (2016) utilized this method in implementing an SMT-based solver for such programs. In the conclusion of this work, we rely on the concept of level ranking by Niemelä (2008) to develop a translation for non-tight constraint answer set programs to SMT formulas so that an SMT solver can be used to compute answer sets of such programs.

Paper outline: Section 2 is on preliminaries. It reviews concepts of logic programs, completion, (input) answer sets, and level ranking. Section 2.2 presents the details of generalized constraint satisfaction problem (GCSP) and links this notion to classical constraint satisfaction. The section on preliminaries concludes with formal definitions of linear and integer linear constraints. Section 3 introduces constraint answer set programs and constraint formulas. Next, in Section 4, we present SMT and specify a class of uniform theories. Section 5 defines SMT formulas and ASPT programs. Uniform theories provide us with a ground to establish a formal link between CASP and SMT in Section 6 by relating SMT formulas with constraint formulas and ASPT programs with constraint answer set programs. Section 6 concludes by characterizing a family of distinct CASP formalisms using the uniform terminology proposed in this work. Section 7 utilizes the generalization of level ranking to propose a method of using SMT solvers for computing answer sets of constraint answer set programs with integer constraints. Finally, we list the conclusions.

2 Preliminaries

This section starts by reviewing logic programs and the concept of an answer set. It also introduces programs' completion. Next, the GCSPs are introduced and related to the classical constraint satisfaction problems (CSPs) studied in artificial intelligence.

2.1 Logic programs and completion

Syntax: A *vocabulary* is a set of propositional symbols also called atoms. As customary, a *literal* is an atom a or its negation, denoted $\neg a$. A *(propositional) logic program*, denoted by Π, over vocabulary σ is a set of *rules* of the form

$$a \leftarrow b_1, \ldots, b_\ell, \ not\ b_{\ell+1}, \ldots, \ not\ b_m, \ \ not\ \ not\ b_{m+1}, \ldots, \ not\ \ not\ b_n \tag{2.1}$$

where a is an atom over σ or $\bot$, and each b_i, $1 \leqslant i \leqslant n$, is an atom in σ. We will sometimes use the abbreviated form for rule (2.1)

$$a \leftarrow B \tag{2.2}$$

where B stands for $b_1, \ldots, b_\ell, \ not\ b_{\ell+1}, \ldots, \ not\ b_m, \ not\ \ not\ b_{m+1}, \ldots, \ not\ \ not\ b_n$ and is also called a *body*. Syntactically, we identify rule (2.1) with the propositional formula

$$b_1 \wedge \ldots \wedge b_\ell \wedge \neg b_{\ell+1} \wedge \ldots \wedge \neg b_m \wedge \neg\neg b_{m+1} \wedge \ldots \wedge \neg\neg b_n \rightarrow a \tag{2.3}$$

and B with the propositional formula

$$b_1 \wedge \ldots \wedge b_\ell \wedge \neg b_{\ell+1} \wedge \ldots \wedge \neg b_m \wedge \neg\neg b_{m+1} \wedge \ldots \wedge \neg\neg b_n. \tag{2.4}$$

Note that (i) the order of terms in (2.4) is immaterial, (ii) *not* is replaced with classical negation ($\neg$), and (iii) comma is replaced with conjunction ($\wedge$). Expression

$$b_1 \wedge \ldots \wedge b_\ell$$

in formula (2.4) is referred to as the *positive* part of the body and the remainder of (2.4) as the *negative* part of the body. Sometimes, we interpret semantically rule (2.1) and its body as propositional formulas, in these cases it is obvious that double negation $\neg\neg$ in (2.3) and (2.4) can be dropped.

The expression a is the *head* of the rule. When a is $\bot$, we often omit it and say that the head is empty. We write $hd(\Pi)$ for the set of non-empty heads of rules in Π.

We call a rule whose body is empty a *fact*. In such cases, we drop the arrow. We sometimes may identify a set X of atoms with the set of facts $\{a. \mid a \in X\}$.

Semantics: We say a set X of atoms *satisfies* rule (2.1), if X satisfies the propositional formula (2.3), where we identify X with an assignment over the atoms in (2.3) in a natural way:

- any atom that occurs in X maps to truth value *true*, and
- any atom in (2.3) but not in X maps to truth value *false*.

We say X satisfies a program Π, if X satisfies every rule in Π. In this case, we also say that X is a model of Π. We may abbreviate satisfaction relation with symbol $\models$ (to denote that a set of atoms satisfies a rule or a program or a formula).

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

The *reduct* Π^X of a program Π relative to a set X of atoms is obtained by first removing all rules (2.1) such that X does not satisfy negative part of the body $\neg b_{\ell+1} \wedge \ldots \wedge \neg b_m \wedge \neg\neg b_{m+1} \wedge \ldots \wedge \neg\neg b_n$, and replacing all remaining rules with $a \leftarrow b_1, \ldots, b_\ell$. A set X of atoms is an *answer set*, if it is the minimal set that satisfies all rules of Π^X (Lifschitz *et al.* 1999).

Ferraris and Lifschitz (2005) showed that a choice rule $\{a\} \leftarrow B$ can be seen as an abbreviation for a rule $a \leftarrow \ not\ not\ a, B$ (choice rules were introduced by Niemelä and Simons (2000) and are commonly used in answer set programming languages). We adopt this abbreviation in the rest of the paper.

Example 1
Consider the logic program from Balduccini and Lierler (2017):

$$\begin{array}{l} \{switch\}. \\ lightOn \leftarrow \ switch, not\ am. \\ \leftarrow not\ lightOn. \\ \{am\}. \end{array} \tag{2.5}$$

Each rule in the program can be understood as follows:

- The action *switch* is exogenous.
- The light is on (*lightOn*) during the night (*not am*) when the action *switch* has occurred.
- The light must be on.
- It is night (*not am*) or morning (*am*)

Choice rules $\{switch\}.$ and $\{am\}.$ in program (2.5) abbreviate rules

$$\begin{array}{l} switch \leftarrow not\ not\ switch. \\ am \leftarrow not\ not\ am. \end{array}$$

respectively. Consider set $\{switch,\ lightOn\}$ of atoms. The reduct of program (2.5) relative to this set follows:

$$\begin{array}{l} switch. \\ lightOn \leftarrow \ switch. \end{array}$$

It is easy to see that set $\{switch,\ lightOn\}$ satisfies every rule of the reduct. Furthermore, this set is minimal among sets with this property. Thus, it is an answer set of program (2.5). In fact, it is the only answer set of this program. This answer set suggests that the only situation that satisfies the specifications of the problem is such that (i) it is currently night, (ii) the light has been switched on, and (iii) the light is on.

Completion: It is customary for a given vocabulary σ, to identify a set X of atoms over σ with (i) a complete and consistent set of literals over σ constructed as $X \cup \{\neg a \mid a \in \sigma \setminus X\}$, and respectively with (ii) an assignment function or interpretation that assigns truth value *true* to every atom in X and *false* to every atom in $\sigma \setminus X$.

By $Bodies(\Pi, a)$ we denote the set of the bodies of all rules of Π with head a. For a program Π over vocabulary σ, the *completion* of Π (Clark 1978), denoted by

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

$Comp(\Pi)$, is the set of classical formulas that consists of the rules (2.1) in Π (recall that we identify rule (2.1) with implication (2.3)) and the implications

$$a \rightarrow \bigvee_{a \leftarrow B \in \Pi} B \tag{2.6}$$

for all atoms a in σ. When set $Bodies(\Pi, a)$ is empty, the implication (2.6) has the form $a \rightarrow \bot$. When a rule (2.2) is a fact a, we identify this rule with the clause consisting of a single atom a.

Example 2
The completion of logic program (2.5) consists of formulas

$$\begin{array}{l} \neg\neg switch \rightarrow switch, \\ switch \wedge \neg am \rightarrow lightOn, \\ \neg lightOn \rightarrow \bot, \\ \neg\neg am \rightarrow am, \\ switch \rightarrow \neg\neg switch, \\ lightOn \rightarrow switch \wedge \neg am, \\ am \rightarrow \neg\neg am. \end{array} \tag{2.7}$$

It is easy to see that this completion is equivalent to the set of formulas

$$\begin{array}{l} lightOn \leftrightarrow switch \wedge \neg am, \\ lightOn. \end{array} \tag{2.8}$$

The set $\{switch, lightOn\}$ is the only model of (2.8). (Unless the signature of a formula is explicitly stated, we consider the set of atoms occurring in the formula to implicitly specify its signature.) Note that set $\{switch, lightOn\}$ coincides with the only answer set of program (2.5).

Tightness: Any answer set of a program is also a model of its completion. The converse does not always hold. Yet, for the large class of logic programs, called *tight*, their answer sets coincide with models of their completion (Fages 1994; Erdem and Lifschitz 2001). Tightness is a syntactic condition on a program that can be verified by means of program's dependency graph.

The *dependency graph* of Π is the directed graph G such that

- the vertices of G are the atoms occurring in Π, and
- for every rule (2.1) in Π whose head is not $\bot$, G has an edge from atom a to each atom in positive part $b_1, \ldots, b_\ell$ of its body.

A program is called *tight* if its dependency graph is acyclic.

For example, the dependence graph of program (2.5) consists of three nodes, namely, *am*, *switch*, and *lightOn* and a single edge from *lightOn* to *switch*. This program is obviously tight.

Level rankings: Niemelä (2008) characterized answer sets of "normal" logic programs in terms of "level rankings." Normal programs consist of rules of the form (2.1), where $n = m$ and a is an atom. Thus, such constructs as choice rules and so-called denials (rules with empty head) are not covered by normal programs. We

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

generalize the concept of level ranking to programs considered in this paper that are more general than normal ones.

We start by introducing some notation. By $\mathbb{N}$ we denote the set of natural numbers. For a rule (2.2), by B^+ we denote its positive part and sometimes identify it with the set of atoms that occur in it, i.e., $\{b_1, \ldots, b_l\}$ (recall that B in (2.2) stands for the right-hand side of the arrow in rule (2.1)). For a program Π, by $At(\Pi)$ we denote the set of atoms occurring in it.

Definition 1
For a logic program Π and a set X of atoms over $At(\Pi)$, a function $\mathrm{lr}\colon X \to \mathbb{N}$ is a *level-ranking* of X for Π when for each $a \in X$, there is B in $Bodies(\Pi, a)$ such that X satisfies B and for every $b \in B^+$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$.

Niemelä (2008) observed that for an arbitrary normal logic programs, a model X of its completion is also an answer set for this program when there is a level ranking of X for the program. We generalize this result beyond normal programs.

Theorem 1
For a program Π and a set X of atoms that is a model of its completion $Comp(\Pi)$, X is an answer set of Π if and only if there is a level ranking of X for Π.

Proof
The proof largely follows the lines of the proof of Theorem 1 from Niemelä (2008) but utilizes the terminology used in this paper. We start by defining an operator $T_\Pi(I)$ for a program Π and a set I over $At(\Pi) \cup \{\bot\}$ as follows:

$$T_\Pi(I) = \{a \mid a \leftarrow B \in \Pi, I \text{ satisfies } B\}.$$

For this operator, we define

$$T_\Pi \uparrow 0 = \emptyset$$

and for $i = 0, 1, 2, \ldots$

$$T_\Pi \uparrow (i+1) = T_\Pi(T_\Pi \uparrow i)$$

Left-to-right: Assume X is an answer set of Π. We can construct a level ranking lr of X for Π using the $T_{\Pi^X}(\cdot)$ operator. As X is an answer set of Π, we know that $X = T_{\Pi^X} \uparrow \omega$ and for each $a \in X$ there is a unique i such that $a \in T_{\Pi^X} \uparrow i$, but $a \notin T_{\Pi^X} \uparrow (i-1)$. We consider $\mathrm{lr}(a) = i$. We now illustrate that lr is indeed a level ranking. For $a \in X$, there is a rule $a \leftarrow B$ of the form (2.1) such that $a \leftarrow b_1, \ldots, b_\ell \in \Pi^X$ and $T_{\Pi^X} \uparrow (i-1)$ satisfies $b_1 \wedge \ldots \wedge b_\ell$. Consequently, for every b_j in $\{b_1, \ldots, b_\ell\}$, $lr(b_j) \leqslant i-1$. Thus, $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b_j)$. Also, from the way the reduct is constructed it follows that X satisfies body B of rule $a \leftarrow B$.

Right-to-left: Assume that there is a level ranking lr of X for Π. We show that then X is the minimal model of Π^X, which implies that X is an answer set of Π. Since X is a model of the completion of Π it follows that X is also a model of Π. Indeed, recall the construction process of completion. From the construction of Π^X it follows that X is also a model of Π^X.

Proof by contradiction: Assume that there is a set of atoms $X' \subset X$ such that X' is a model of Π^X, and hence X is not a minimal model of Π^X. Consider now an atom

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Table 1. *Example definitions for signature, valuation, and r-denotation*

Σ_1	$\{s, r, E, Q\}$
D_1	$\{1, 2, 3\}$
v_1	$[\Sigma_{1\|v}, D_1]$ valuation, where $s^{v_1} = 1$ and $r^{v_1} = 1$
v_2	$[\Sigma_{1\|v}, D_1]$ valuation, where $s^{v_2} = 2$ and $r^{v_2} = 1$
ρ_1	$[\Sigma_{1\|r}, D_1]$ r-denotation, where $E^{\rho_1} = \{\langle 1\rangle\}$, $Q^{\rho_1} = \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle\}$
ρ_2	$[\Sigma_{1\|r}, D_1]$ r-denotation, where $E^{\rho_2} = \{\langle 2\rangle, \langle 3\rangle\}$, $Q^{\rho_2} = Q^{\rho_1}$.

$a \in X \setminus X'$ with the smallest level ranking $\mathrm{lr}(a)$. Since lr is a level ranking of X for Π, it follows that there is a rule $a \leftarrow B$ in Π such that $X \models B$ and for every $b \in B^+$, it holds that $\mathrm{lr}(b) < \mathrm{lr}(a)$. As we considered atom a in $X \setminus X'$ with the smallest level ranking it follows that $b \in X'$. By Π^X construction, rule $a \leftarrow B^+$ belongs to Π^X. We derive that X' satisfies B^+, but does not contain a. This contradicts out assumption that X' satisfies Π^X as it does not satisfy rule $a \leftarrow B^+$. □

2.2 Generalized constraint satisfaction problems

In this section, we present a primitive constraint as defined by Marriott and Stuckey (1998, Section 1.1). We refer to this concept as a constraint, dropping the word "primitive." We use constraints to define a GCSP that Marriott and Stuckey refer to as "constraint." We then review CSPs as commonly defined in artificial intelligence literature and illustrate that they form a special case of GCSPs. We finally introduce linear constraints and linear CSPs.

Signature, c-vocabulary, constraint atoms: We adopt the following convention: for a function v and an element x, by x^v we denote the value that function v maps x to, in other words, $x^v = v(x)$. A *domain* is a *non-empty* set of elements (or values). A *signature* Σ is a set of *variables*, *predicate symbols*, and *function symbols (or f-symbols)*. Predicate and function symbols are associated with a positive integer called *arity*. By $\Sigma_{|v}$, $\Sigma_{|r}$, and $\Sigma_{|f}$ we denote the subsets of Σ that contain all variables, all predicate symbols, and all f-symbols, respectively.

For instance, we can define signature $\Sigma_1 = \{s, r, E, Q\}$ by saying that s and r are variables, E is a predicate symbol of arity 1, and Q is a predicate symbol of arity 2. Then, $\Sigma_{1|v} = \{s, r\}$, $\Sigma_{1|r} = \{E, Q\}$, $\Sigma_{1|f} = \emptyset$.

Let D be a domain. For a set V of variables, we call a total function $v : V \to D$ a $[V, D]$ *valuation*. For a set R of predicate symbols, we call a total function on R *an* $[R, D]$ *r-denotation*, when it maps each n-ary predicate symbol of R into an n-ary relation on D. For a set F of f-symbols, we call a total function on F *an* $[F, D]$ *f-denotation*, when it maps each n-ary f-symbol of F into a function $D^n \to D$.

Table 1 presents sample definitions of a domain, valuations, and r-denotations. In the remainder of the paper, we frequently refer to these sample valuations, and r-denotations.

A *constraint vocabulary (c-vocabulary)* is a pair $[\Sigma, D]$, where Σ is a signature and D is a domain. A *term* over a c-vocabulary $[\Sigma, D]$ is either

- a variable in $\Sigma_{|v}$,
- a domain element in D, or

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

- an expression $f(t_1,\ldots,t_n)$, where f is an f-symbol of arity n in $\Sigma_{|f}$ and $t_1,\ldots,t_n$ are terms over $[\Sigma, D]$.

A *constraint atom* over a c-vocabulary $[\Sigma, D]$ is an expression

$$P(t_1,\ldots,t_n), \tag{2.9}$$

where P is a predicate symbol from $\Sigma_{|r}$ of arity n and $t_1,\ldots,t_n$ are terms over $[\Sigma, D]$. A *constraint literal* over a c-vocabulary $[\Sigma, D]$ is either a constraint atom (2.9) or an expression

$$\neg P(t_1,\ldots,t_n), \tag{2.10}$$

where $P(t_1,\ldots,t_n)$ is a constraint atom over $[\Sigma, D]$.

For instance, expressions

$$\neg E(s), \quad \neg E(2), \quad Q(r,s)$$

are constraint literals over c-vocabulary $[\Sigma_1, D_1]$, where Σ_1 and D_1 are defined in Table 1.

It is due to notice that syntactically, constraint literals are similar to *ground* literals of predicate logic. (In predicate logic, variables as defined here are referred to as *object constants* or *function symbols of arity* 0.) The only difference is that here domain elements are allowed to form a term. For instance, an expression $E(2)$ is a constraint atom over $[\Sigma_1, D_1]$, where 2 is a term formed from a domain element. In predicate logic, domain elements are not part of a signature over which atoms are formed.

We now proceed to introducing satisfaction relation for constraint literals. Let $[\Sigma, D]$ be a c-vocabulary, v be a $[\Sigma_{|v}, D]$ valuation, ρ be a $[\Sigma_{|r}, D]$ r-denotation, and ϕ be a $[\Sigma_{|f}, D]$ f-denotation. First, we define recursively a value that valuation v assigns to a term τ over $[\Sigma, D]$ with respect to ϕ. We denote this value by $\tau^{v,\phi}$ and compute it as follows:

- for a term that is a variable x in $\Sigma_{|v}$, $x^{v,\phi} = x^v$,
- for a term that is a domain element d in D, $d^{v,\phi}$ is d itself,
- for a term τ of the form $f(t_1,\ldots,t_n)$, $\tau^{v,\phi}$ is defined recursively by the formula

$$f(t_1,\ldots,t_n)^{v,\phi} = f^{\phi}(t_1^{v,\phi},\ldots,t_n^{v,\phi}).$$

Second, we define what it means for valuation to be a solution of a constraint literal with respect to given r- and f-denotations. We say that v *satisfies (is a solution to)* constraint literal (2.9) over $[\Sigma, D]$ *with respect to* ρ *and* ϕ when $\langle t_1^{v,\phi},\ldots,t_n^{v,\phi}\rangle \in P^{\rho}$. Let $\mathscr{R}$ be an n-ary relation on D. By $\overline{\mathscr{R}}$ we denote *complement* relation of $\mathscr{R}$ constructed as $D^n \setminus \mathscr{R}$. Valuation v *satisfies (is a solution to)* constraint literal of the form (2.10) *with respect to* ρ *and* ϕ when $\langle t_1^{v,\phi},\ldots,t_n^{v,\phi}\rangle \in \overline{P^{\rho}}$.

For instance, consider declarations of valuations v_1 and v_2, and r-denotations ρ_1 and ρ_2 in Table 1. Valuation v_1 satisfies constraint literal $Q(r,s)$ with respect to ρ_1, while valuation v_2 does not satisfy this constraint literal with respect to ρ_2.

Lexicon, constraints, generalized constraint satisfaction problem: We are now ready to define constraints, their syntax and semantics. To begin we introduce a *lexicon*,

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Table 2. *Sample lexicons and constraints*

$\mathscr{L}_1$	$([\Sigma_1, D_1], \rho_1)$
$\mathscr{L}_2$	$([\Sigma_1, D_1], \rho_2)$
c_1	a literal $Q(r, s)$ over lexicon $\mathscr{L}_1$
c_2	a literal $Q(r, s)$ over lexicon $\mathscr{L}_2$
c_3	a literal $\neg E(s)$ over lexicon $\mathscr{L}_2$
c_4	a literal $\neg E(2)$ over lexicon $\mathscr{L}_2$.

which is a tuple $([\Sigma, D], \rho, \phi)$, where $[\Sigma, D]$ is a c-vocabulary, ρ is a $[\Sigma_{|r}, D]$ r-denotation, and ϕ is a $[\Sigma_{|f}, D]$ f-denotation. For a lexicon $\mathscr{L} = ([\Sigma, D], \rho, \phi)$, we call any function that is $[\Sigma_{|v}, D]$ valuation, a *valuation over* $\mathscr{L}$. We omit the last element of the lexicon tuple if the signature Σ of the lexicon contains no f-symbols. A *constraint* is defined over lexicon $\mathscr{L} = ([\Sigma, D], \rho, \phi)$. Syntactically, it is a constraint literal over $[\Sigma, D]$ (lexicon $\mathscr{L}$, respectively). Semantically, we say that valuation v over $\mathscr{L}$ *satisfies (is a solution to)* the constraint c when v satisfies c with respect to ρ and ϕ.

For instance, Table 2 presents definitions of sample lexicons $\mathscr{L}_1$, $\mathscr{L}_2$, and constraints c_1, c_2, c_3, and c_4 using the earlier declarations from Table 1. Valuation v_1 from Table 1 is a solution to c_1, c_2, c_3, but not a solution to c_4. Valuation v_2 from Table 1 is not a solution to c_1, c_2, c_3, and c_4. In fact, constraint c_4 has no solutions. We sometimes omit the explicit mention of the lexicon when talking about constraints: we then may identify a constraint with its syntactic form of a constraint literal.

Definition 2
A *GCSP* $\mathscr{C}$ is a finite set of constraints over a lexicon $\mathscr{L} = ([\Sigma, D], \rho, \phi)$. We say that a valuation v over $\mathscr{L}$ *satisfies (is a solution to)* the GCSP $\mathscr{C}$ when v is a solution to every constraint in $\mathscr{C}$.

For example, consider a set $\{c_2, c_3, c_4\}$ of constraints. Any subset of this set forms a GCSP, including subsets $\{c_2, c_3\}$ and $\{c_2, c_3, c_4\}$. Sample valuation v_1 over lexicon $\mathscr{L}_2$ (where v_1 stems from Tables 1) satisfies the GCSP $\{c_2, c_3\}$, but does not satisfy the GCSP $\{c_2, c_3, c_4\}$.

From GCSP to constraint satisfaction problem: We now define a CSP as customary in classical literature on artificial intelligence. We then explain in which sense GCSPs generalize CSPs.

We say that a lexicon is *finite-domain* if it is defined over a c-vocabulary that refers to a domain whose set of elements is finite. Trivially, lexicons $\mathscr{L}_1$ and $\mathscr{L}_2$ defined in Table 2 are finite-domain lexicons. Consider a special case of a constraint of the form (2.9) over finite-domain lexicon $\mathscr{L} = ([\Sigma, D], \rho)$, so that each t_i is a variable. (For instance, constraints c_1, c_2, and c_3 satisfy the stated requirements, while c_4 does not.) In this case, we can identify constraint (2.9) over $\mathscr{L}$ with the pair

$$\langle (t_1, \ldots, t_n), P^\rho \rangle. \tag{2.11}$$

A *CSP* is a set of pairs (2.11), where $\Sigma_{|v}$ and D of the finite-domain lexicon $\mathscr{L}$ are called the variables and the domain of CSP, respectively. Saying that valuation v

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

over $\mathscr{L}$ satisfies (2.9) is the same as saying that $\langle t_1^v, \ldots, t_n^v \rangle \in P^\rho$. The latter is the way in which a solution to expressions (2.11) in CSP is typically defined. As in the definition of semantics of GCSP, a valuation is a *solution* to a CSP problem C when it is a solution to every pair (2.11) in C.

In conclusion, GCSP generalizes CSP by

- elevating the finite-domain restriction, and
- allowing us more elaborate syntactic expressions (e.g., recall f-symbols).

2.2.1 Linear and integer linear constraints

We now define "numeric" signatures and lexicons and introduce a set of constraints referred to as linear, which are commonly used in practice. A *numeric signature* is a signature that satisfies the following requirements:

- its only predicate symbols are $<, >, \leqslant, \geqslant, =, \neq$ of arity 2, and
- its only f-symbols are $+, \times$ of arity 2.

We use the symbol $\mathscr{A}$ to denote a numeric signature.

Symbols $\mathbb{Z}$ and $\mathbb{R}$ denote the sets of integers and real numbers, respectively. Let $\rho_{\mathbb{Z}}$ and $\phi_{\mathbb{Z}}$ be $[\{<, >, \leqslant, \geqslant, =, \neq\}, \mathbb{Z}]$ r-denotation and $[\{+, \times\}, \mathbb{Z}]$ f-denotation, respectively, where they map their predicate and function symbols into usual arithmetic relations and operations over integers. Similarly, $\rho_{\mathbb{R}}$ and $\phi_{\mathbb{R}}$ denote $[\{<, >, \leqslant, \geqslant, =, \neq\}, \mathbb{R}]$ r-denotation and $[\{+, \times\}, \mathbb{R}]$ f-denotation, respectively, defined over the reals. We can now define the following lexicons:

- an *integer lexicon* of the form $([\mathscr{A}, \mathbb{Z}], \rho_{\mathbb{Z}}, \phi_{\mathbb{Z}})$,
- a *numeric lexicon* of the form $([\mathscr{A}, \mathbb{R}], \rho_{\mathbb{R}}, \phi_{\mathbb{R}})$.

A *(numeric) linear expression* has the form

$$a_1 x_1 + \cdots + a_n x_n, \tag{2.12}$$

where $a_1, \ldots, a_n$ are real numbers and $x_1, \ldots, x_n$ are variables over real numbers. When $a_i = 1$ $(1 \leqslant i \leqslant n)$, we may omit it from the expression. We view expression (2.12) as an abbreviation for the following term

$$+(\times(a_1, x_1), +(\times(a_2, x_2), \cdots + (\times(a_{n-1}, x_{n-1}), \times(a_n, x_n)) \ldots),$$

over some c-vocabulary $[\mathscr{A}, \mathbb{R}]$, where $\mathscr{A}$ contains $x_1, \ldots, x_n$ as its variables. For instance, $2x_2 + 3x_3$ is an abbreviation for the expression $+(\times(2, x_2), \times(3, x_3))$.

An *integer linear expression* has the form (2.12), where $a_1, \ldots, a_n$ are integers, and $x_1, \ldots, x_n$ are variables over integers.

We call a constraint *linear (integer linear)* when it is defined over some numeric (integer) lexicon and has the form

$$\bowtie (e, k) \tag{2.13}$$

where e is a linear (integer linear) expression, k is a real number (an integer), and $\bowtie$ belongs to $\{<, >, \leqslant, \geqslant, =, \neq\}$. We can write (2.13) as an expression in usual infix notation $e \bowtie k$.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

We call a GCSP a *(integer) linear CSP* when it is composed of (integer) linear constraints. For instance, consider integer linear CSP composed of two constraints $x > 4$ and $x < 5$ (here signature $\mathscr{A}$ is implicitly defined by restricting its variable to contain x). It is easy to see that this problem has no solutions. On the other hand, linear CSP composed of the same two constraints $x > 4$ and $x < 5$ has an infinite number of solutions, including valuation that assigns x to 4.1.

3 Constraint answer set programs and constraint formulas

In this section, we introduce constraint answer set programs, which merge the concepts of logic programming and GCSPs. We also present a similar concept of "constraint formulas," which merges the concepts of propositional formulas and GCSPs. First, we introduce input answer sets, followed by constraint answer set programs and input completion. Second, we present constraint formulas. Finally, we demonstrate the close connection between the constraint answer set programs and constraint formulas using the concept of input completion and tightness condition.

3.1 Constraint answer set programs

We start by introducing a concept in spirit of an input answer set by Lierler and Truszczynski (2011).[1] In particular, we consider input answer sets "relative to input vocabularies." We then extend the definition of completion and restate the result by Erdem and Lifshitz (2001) for the case of input answer sets. The concept of an input answer set is essential for introducing constraint answer set programs. Constraint answer set programs (and constraint formulas) are defined over two disjoint vocabularies so that atoms stemming from those vocabularies "behave" differently. Input answer set semantics allows us to account for these differences.

Definition 3
For a logic program Π over vocabulary σ, a set X of atoms over σ is an *input answer set* of Π relative to vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$ when X is an answer set of the program

$$\Pi \cup (X \cap \iota).$$

To illustrate the concept of an input answer set consider program

$$lightOn \leftarrow switch, not\ am.$$
$$\leftarrow not\ lightOn.$$

This program has a unique input answer set $\{switch, lightOn\}$ relative to input vocabulary $\{switch, am\}$.

Let σ_r and σ_i be two disjoint vocabularies. We refer to their elements as *regular* and *irregular* atoms, respectively.

[1] Similar concepts to input answer sets have been noted by Gelfond and Przymusinska (1996), Oikarinen and Janhunen (2006), and Denecker and Vennekens (2007).

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Definition 4
A *constraint answer set program* (CAS program) over the vocabulary $\sigma = \sigma_r \cup \sigma_i$ is a triple $\langle \Pi, \mathscr{B}, \gamma \rangle$, where

- Π is a logic program over the vocabulary σ such that $hd(\Pi) \cap \sigma_i = \emptyset$,
- $\mathscr{B}$ is a set of constraints over some lexicon $\mathscr{L}$, and
- γ is an injective function from the set σ_i of irregular atoms to the set $\mathscr{B}$ of constraints.

For a CAS program $P = \langle \Pi, \mathscr{B}, \gamma \rangle$ over the vocabulary $\sigma = \sigma_r \cup \sigma_i$ so that $\mathscr{L}$ is the lexicon of the constraints in $\mathscr{B}$, a set $X \subseteq At(\Pi)$ is an *answer set* of P if

(1) X is an input answer set of Π relative to σ_i, and
(2) the following GCSP over $\mathscr{L}$ has a solution

$$\{\gamma(a) \mid a \in X \cap \sigma_i\} \cup \{\neg\gamma(a) \mid a \in (At(\Pi) \cap \sigma_i) \setminus X\}. \tag{3.1}$$

Note that $\neg\gamma(a)$ may result in expression of the form $\neg\neg P(t_1, \ldots, t_n)$ that we identify with $P(t_1, \ldots, t_n)$. (We use this convention across the paper.)

These definitions are generalizations of CAS programs introduced by Gebser *et al.* (2009) as they

- refer to the concept of GCSP in place of CSP in the original definition, and
- allow for more general syntax of logic rules (e.g., choice rules are covered by the presented definition).

This is a good place to note a restriction $hd(\Pi) \cap \sigma_i = \emptyset$ on the form of the rules in a CAS program. This restriction states that an irregular atom may not form a head of a rule. There is a body of research including, for example, the works by Bartholomew and Lee (2012), Lifschitz (2012), and Balduccini (2013) that investigate variants of semantics of *CAS-like* programs, where analogous to irregular atoms are allowed in the heads. Lifting this restriction proves to be non-trivial. The traditional CASP systems, such as CLINGCON or EZCSP, restrict their attention to programs discussed here.

It is worthwhile to remark on the second condition in the definition of an answer set for CAS programs. This condition requires the *existence* of a solution to a constructed GCSP problem, and ignores a form of a particular solution to this GCSP or a possibility of multiple solutions. We now define a concept of an extended answer set that takes a solution to a constructed GCSP problem into account:

Definition 5
For a CAS program $P = \langle \Pi, \mathscr{B}, \gamma \rangle$ over the vocabulary $\sigma = \sigma_r \cup \sigma_i$ so that $\mathscr{L}$ is the lexicon of the constraints in $\mathscr{B}$, a set $X \subseteq \sigma$ and valuation v from variables in the signature of $\mathscr{L}$ to the domain of $\mathscr{L}$, a pair $\langle X, v \rangle$ is an *extended answer set* of P if X is an answer set of P and v is a solution to GCSP (3.1).

CASP systems, such as CLINGCON or EZCSP, allow the user to select whether he is interested in computing answer sets or extended answer sets of a given CAS program. In the rest of the paper, we focus on the notion of an answer set, but

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

generalizing concepts and results introduced later to the notion of extended answer set is not difficult.

In the sequel, we adopt the following notation. To distinguish irregular atoms from the constraints to which these atoms are mapped, we use bars to denote that an expression is an irregular atom. For instance, $|x < 12|$ and $|x \geqslant 12|$ denote irregular atoms. Whereas inequalities (constraints) $x < 12$ and $x \geqslant 12$, respectively, provide natural mappings for these atoms. We assume such natural mappings in this presentation.

Example 3

Let us consider sample constraint answer set program. We first define the integer lexicon $\mathscr{L}_3$:

Σ_2	a numeric signature containing one variable $\{x\}$	(3.2)
$\mathscr{L}_3$	$([\Sigma_2, \mathbb{Z}], \rho_{\mathbb{Z}})$	

Second, we define a CAS program

$$P_1 = \langle \Pi_1, \mathscr{B}_{\mathscr{L}_3}, \gamma_1 \rangle \tag{3.3}$$

over integer lexicon $\mathscr{L}_3$, where

- Π_1 is the program

$$\begin{array}{l} \{switch\}. \\ lightOn \leftarrow \ switch, not\ am. \\ \leftarrow not\ lightOn. \\ \{am\}. \\ \leftarrow not\ am, |x < 12|. \\ \leftarrow am, |x \geqslant 12|. \\ \leftarrow |x < 0|. \\ \leftarrow |x > 23|. \end{array} \tag{3.4}$$

  The set of irregular atoms of Π_1 is $\{|x < 12|, |x \geqslant 12|, |x < 0|, |x > 23|\}$. The remaining atoms form the regular set. The first four lines of program Π_1 are identical to these of logic program (2.5). The last four lines of the program state the following:

  — It must be *am* when $x < 12$, where x is understood as the hours.
  — It is impossible for it to be *am* when $x \geqslant 12$.
  — Variable x must be non-negative.
  — Variable x must be less than or equal to 23.

- $\mathscr{B}_{\mathscr{L}_3}$ is the set of all integer linear constraints over integer lexicon $\mathscr{L}_3$, which obviously includes constraints $\{x < 12, x \geqslant 12, x < 0, x > 23\}$,

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

- $\gamma_1(a) = \begin{cases} \text{constraint } x < 12 \text{ over integer lexicon } \mathscr{L}_3 & \text{if } a = |x < 12| \\ \text{constraint } x \geqslant 12 \text{ over integer lexicon } \mathscr{L}_3 & \text{if } a = |x \geqslant 12|. \\ \text{constraint } x < 0 \text{ over integer lexicon } \mathscr{L}_3 & \text{if } a = |x < 0| \\ \text{constraint } x > 23 \text{ over integer lexicon } \mathscr{L}_3 & \text{if } a = |x > 23|. \end{cases}$

Consider the set

$$\{switch,\ lightOn, |x \geqslant 12|\} \tag{3.5}$$

over $At(\Pi_1)$. This set is the only input answer set of Π_1 relative to its irregular atoms. Also, the integer linear CSP with constraints

$$\{\gamma_1(|x \geqslant 12|), \neg\gamma_1(|x < 12|), \neg\gamma_1(|x < 0|), \neg\gamma_1(|x > 23|)\}$$
$$=$$
$$\{x \geqslant 12, \neg x < 12, \neg x < 0, \neg x > 23\}$$

has a solution. There are 12 valuations $v_1 \ldots v_{12}$ relative to integer lexicon $\mathscr{L}_3$ for x, which satisfy this GCSP: $x^{v_1} = 12, \ldots, x^{v_{12}} = 23$. It follows that set (3.5) is an answer set of P_1. Pair $\langle\{switch,\ lightOn, |x \geqslant 12|\}, v_1\rangle$ is one of the twelve extended answer sets of P_1.

3.1.1 On grounding

In practice, CASP languages similarly to ASP languages, allow for non-constraint variables. Gebser *et al.* (2009) present a program written in the language supported by CASP solver CLINGCON for the so called *bucket* problem. We list a rule from that program to illustrate the notion of non-constraint variables[2]:

```
:- volume(B,T+1) $!= volume(B,T) $+ A, pour(B,T,A),
   bucket(B), time(T), amount(A).
```

Non-constraint variables of these rules are B, T, and A. The sign \$ marks the irregular atoms. Rules of the kind are interpreted as shorthands for the set of rules without non-constraint variables, called *ground rules*. Ground rules are obtained by replacing every non-constraint variable in the rules by suitable terms not containing non-constraint variables. For instance, if suitable terms for non-constraint variable B in the sample rule above range over a single value a; suitable terms for T range over two values 0 and 1; and suitable terms for A range over two values 1 and 2, then the CLINGCON rule above is instantiated as follows:

```
:- volume(a,1) $!= volume(a,0) $+ 1, pour(a,0,1),
   bucket(a), time(0), amount(1).
:- volume(a,2) $!= volume(a,1) $+ 1, pour(a,1,1),
   bucket(a), time(1), amount(1).
:- volume(a,1) $!= volume(a,0) $+ 2, pour(a,0,2),
```

[2] A rule is taken form the site documenting system CLINGCON: `http://www.cs.uni-potsdam.de/clingcon/examples/bucket_torsten.lp` .

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

```
   bucket(a), time(0), amount(2).
:- volume(a,2) $!= volume(a,1) $+ 2, pour(a,1,2),
   bucket(a), time(1), amount(2).
```

To help to map these rules into the syntax used earlier, we rewrite the first one using notation of this paper:

$$\leftarrow |volume(a,1) \neq volume(a,0) + 1|, pour(a,0,1), bucket(a), time(0), amount(1).$$

Expressions $volume(a,0)$, $volume(a,1)$, and $volume(a,2)$ stand for constraint variables in the signature of integer lexicon of this program. System CLINGCON supports integer linear constraints, where a constraint atom $|volume(a,1) \neq volume(a,0) + 1|$ is naturally mapped into integer linear constraint $volume(a,1) \neq volume(a,0) + 1$.

The process of replacing non-ground rules by their ground counterparts is called *grounding* and is well understood in ASP (Gebser *et al.* 2007; Calimeri *et al.* 2008). The availability of non-constraint variables makes modeling in CASP an attractive and easy process. In fact, this feature distinguishes greatly CASP and SMT paradigms. An SMT-LIB language (Barrett *et al.* 2015) is a standard language for interfacing major SMT solvers. This language does not provide a user with convenient modeling capabilities. There is an underlying assumption that code in SMT-LIB is to be generated by special-purpose programs.

Since grounding is a standalone process in CASP and a non-ground program is viewed as a shorthand for ground programs, the focus of this paper is on the later.

3.1.2 Input completion

Similar to how completion was defined in Section 2, we now define an input completion that is relative to an (input) vocabulary.

Definition 6
For a program Π over vocabulary σ, the *input-completion* of Π relative to vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$, denoted by $IComp(\Pi, \iota)$, is defined as the set of formulas in propositional logic that consists of the rules (2.3) in Π and the implications (2.6) for all atoms a occurring in $\sigma \setminus \iota$.

Example 4
Here, we illustrate the concept of input completion. Consider program Π_1 from Example 3. Its input completion relative to a vocabulary consisting of its irregular atoms $\{|x < 12|, |x \geqslant 12|, |x < 0|, |x > 23|\}$ consists of formulas in (2.7) and formulas in

$$\begin{array}{l} \neg am \wedge |x < 12| \rightarrow \bot \\ am \wedge |x \geqslant 12| \rightarrow \bot. \\ |x < 0| \rightarrow \bot. \\ |x > 23| \rightarrow \bot. \end{array} \tag{3.6}$$

It is easy to see that $IComp(\Pi_1, \{|x < 12|, |x \geqslant 12|, |x < 0|, |x > 23|\})$ is equivalent to the union of (2.8) and (3.6). The set $\{switch, lightOn, |x \geqslant 12|\}$ is the only model of this input completion. Note that this model coincides with the input answer set of Π_1 relative to the set of its irregular atoms.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

The observation that we made last in the preceding example is an instance of the general fact captured by the following theorem.

Theorem 2
For a tight program Π over vocabulary σ and vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$, a set X of atoms from σ is an input answer set of Π relative to ι if and only if X satisfies the program's input-completion $IComp(\Pi, \iota)$.

Furthermore, for any program any of its input answer sets is also a model of its input-completion.

Theorem 3
For a program Π over vocabulary σ and vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$, if a set X of atoms from σ is an input answer set of Π relative to ι, then X satisfies the program's input-completion $IComp(\Pi, \iota)$.

To prove these theorems, it is useful to state the following lemma.

Lemma 1
For a program Π over vocabulary σ and vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$, a set X of atoms over σ is a model of formula $IComp(\Pi, \iota)$ if and only if it is a model of $Comp(\Pi \cup (X \cap \iota))$.

Proof
Since we identify rules (2.2) in Π with respective implications $B \rightarrow a$, we may write symbol Π to denote not only a set of rules but also a set of respective implications.

We can write $IComp(\Pi, \iota)$ as the union of

$$\Pi, \tag{3.7}$$

implications

$$\{a \rightarrow \bigvee_{B \in Bodies(\Pi, a)} B \mid a \in hd(\Pi)\}, \tag{3.8}$$

and

$$\{a \rightarrow \bot \mid a \notin hd(\Pi) \text{ and } a \in \sigma \setminus \iota\}. \tag{3.9}$$

Note that expression (3.9) can be written as

$$\{a \rightarrow \bot \mid a \notin hd(\Pi) \text{ and } a \notin \iota \text{ and } a \in \sigma\}. \tag{3.10}$$

Similarly, we can write $Comp(\Pi \cup (X \cap \iota))$ as a union of (3.7),

$$(X \cap \iota) \tag{3.11}$$

$$\{a \rightarrow \bigvee_{B \in Bodies(\Pi \cup (X \cap \iota), a)} B \mid a \in hd(\Pi)\} \tag{3.12}$$

$$\{a \rightarrow \bot \mid a \notin hd(\Pi) \text{ and } a \notin (X \cap \iota) \text{ and } a \in \sigma\} \tag{3.13}$$

Left-to-right: Assume $X \models IComp(\Pi, \iota)$. It consists of (3.7), (3.8), and (3.10) by construction. Trivially, X satisfies (3.11). Thus, we are left to show that X

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

satisfies (3.12) and (3.13). Note that (3.8) and (3.12) coincide since $Bodies(\Pi, a) = Bodies(\Pi \cup (X \cap \iota)), a)$ for all atoms $a \in hd(\Pi)$ as $hd(\Pi) \cap \iota = \emptyset$. Consequently, X satisfies (3.12). Observe set (3.13) can be written as the union of set (3.10) and set

$$\{a \rightarrow \bot \mid a \notin hd(\Pi) \text{ and } a \in (\iota \setminus X) \text{ and } a \in \sigma\}. \tag{3.14}$$

Trivially, X satisfies (3.14) as any atom that satisfies condition $a \in (\iota \setminus X)$ is such that $a \notin X$. It also satisfies (3.10). Consequently, X satisfies (3.13).

Right-to-left: Assume X is a model of $Comp(\Pi \cup (X \cap \iota))$. Set X satisfies (3.7) and (3.11)–(3.13). Since X satisfies (3.12), it satisfies (3.8) (as they coincide as argued above). Since X satisfies (3.13), X also satisfies (3.10) (recall set (3.13) can be written as the union of set (3.10) and set (3.14)). Consequently, $X \models ICom p(\Pi, \iota)$. □

Proof of Theorem 2
We are given that Π is tight. Since $X \cap \iota$ only consists of facts, it follows that $\Pi \cup (X \cap \iota)$ is tight also.

Left-to-right: Assume X is an input answer set of a program Π relative to ι. By Definition 3, X is an answer set of $\Pi \cup (X \cap \iota)$. It is well known that an answer set of a program is also a model of its completion. Thus, X is a model of $Comp(\Pi \cup (X \cap \iota))$. By Lemma 1, X is a model of $ICom p(\Pi, \iota)$.

Right-to-left: Assume $X \models ICom p(\Pi, \iota)$. By Lemma 1, X is a model of completion $Comp(\Pi \cup (X \cap \iota))$. By results by Fages (1994) and Erdem and Lifshitz (2001), X is the answer set of the program $\Pi \cup (X \cap \iota)$. By Definition 3, X is an input answer set of Π relative to ι. □

Direction Left-to-right of the proof of Theorem 2 serves as a proof for Theorem 3 also.

3.2 Constraint formula

Just as we defined constraint answer set programs, we can define constraint formulas.

For a propositional formula F, by $At(F)$ we denote the set of atoms occurring in it.

Definition 7
A *constraint formula* over the vocabulary $\sigma = \sigma_r \cup \sigma_i$ is a triple $\langle F, \mathscr{B}, \gamma \rangle$, where

- F is a propositional formula over the vocabulary σ,
- $\mathscr{B}$ is a set of constraints over some lexicon $\mathscr{L}$, and
- γ is an injective function from the set σ_i of irregular atoms to the set $\mathscr{B}$ of constraints.

For a constraint formula $\mathscr{F} = \langle F, \mathscr{B}, \gamma \rangle$ over the vocabulary $\sigma = \sigma_r \cup \sigma_i$ such that $\mathscr{L}$ is the lexicon of the constraints in $\mathscr{B}$, a set $X \subseteq At(F)$ is a *model* of $\mathscr{F}$ if

(1) X is a model of F, and
(2) the following GCSP over $\mathscr{L}$ has a solution

$$\{\gamma(a) | a \in X \cap \sigma_i\} \cup \{\neg\gamma(a) | a \in (At(F) \cap \sigma_i) \setminus X\}.$$

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Example 5
Similar to the CAS program P_1 from Example 3, we can define a constraint formula

$$\mathscr{F}_1 = \langle IComp(\Pi_1, \{|x < 12|, |x \geqslant 12|, |x < 0|, |x > 23|\}), \mathscr{B}_{\mathscr{L}_3}, \gamma_1 \rangle$$

relative to integer lexicon $\mathscr{L}_3$. We understand Π_1, $\mathscr{B}_{\mathscr{L}_3}$, and γ_1 as in Example 3. Example 4 illustrates how input completion $IComp(\Pi_1, \{|x < 12|, |x \geqslant 12|, |x < 0|, |x > 23|\})$ is formed. The set

$$\{switch, lightOn, |x \geqslant 12|\}$$

is the only model of $\mathscr{F}_1$.

Following theorem captures a relation between CAS programs and constraint formulas. This theorem is an immediate consequence of Theorem 2.

Theorem 4
For a CAS program $P = \langle \Pi, \mathscr{B}, \gamma \rangle$ over the vocabulary $\sigma = \sigma_r \cup \sigma_i$ and a set X of atoms over σ, when Π is tight, X is an answer set of P if and only if X is a model of constraint formula $\langle IComp(\Pi, \sigma_i), \mathscr{B}, \gamma \rangle$ over $\sigma = \sigma_r \cup \sigma_i$.

We note that Example 3 and Example 5 demonstrate this property. In the future, we will abuse the term "tight." We will refer to CAS program $P = \langle \Pi, \mathscr{B}, \gamma \rangle$ as *tight* when its first member Π has this property.

4 Satisfiability modulo theories

First, in this section, we introduce the notion of a "theory" in SMT (Barrett and Tinelli 2014). Second, we present the definition of a "restriction formula" and state the conditions under which such formulas are satisfied by a given interpretation. These formulas are syntactically restricted classical ground predicate logic formulas. To be precise, a restriction formula corresponds to a conjunction of ground literals. The presented notions of interpretation and satisfaction are usual, but are stated in terms convenient for our purposes.

Definition 8
An *interpretation* I for a signature Σ, or Σ-*interpretation*, is a tuple (D, v, ρ, ϕ) where

- D is a domain,
- v is a $[\Sigma_{|v}, D]$ valuation,
- ρ is a $[\Sigma_{|r}, D]$ r-denotation, and
- ϕ is a $[\Sigma_{|f}, D]$ f-denotation.

For a signature Σ, a Σ-*theory* is a set of interpretations over Σ.

For signatures that contain no f-symbols, we omit the reference to the last element of the interpretation tuple. For instance, for signature Σ_1 from Table 1, consider the following sample interpretations:

$\mathscr{I}_1$	(D_1, v_1, ρ_1)
$\mathscr{I}_2$	(D_1, v_2, ρ_1)

(4.1)

Any subset of interpretations $\{\mathscr{I}_1, \mathscr{I}_2\}$ exemplifies a unique Σ_1-theory.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

As mentioned earlier, in literature on predicate logic and SMT, the terms *object constant* and *function symbol of arity* 0 are commonly used to refer to elements in the signature that we call variables. Here, we use the terms that stem from definitions related to constraint satisfaction processing to facilitate uncovering the precise link between CASP-like formalisms and SMT-like formalisms. It is typical for predicate logic signatures to contain propositional symbols (predicate symbols of arity 0). It is easy to extend the notion of signature introduced here to allow propositional symbols. Yet, it will complicate the presentation, which is the reason we avoid this extension.

A *restriction formula* over a signature Σ is a finite set of constraint literals over a c-vocabulary $[\Sigma, \emptyset]$. A sample restriction formula over Σ_1 follows

$$\{\neg E(s), \neg Q(r, s)\}. \tag{4.2}$$

We now introduce the semantics of restriction formulas. Let $I = (D, v, \rho, \phi)$ be a Σ-interpretation. To each term τ over a c-vocabulary $[\Sigma, \emptyset]$, I assigns a value $\tau^{v,\phi}$ that we denote by τ^I. We say that interpretation I *satisfies* restriction formula Φ over Σ, when v satisfies every constraint literal in Φ with respect to ρ and ϕ. For instance, interpretation $\mathscr{I}_2$ satisfies restriction formula (4.2), while $\mathscr{I}_1$ does not satisfy (4.2).

We say that a restriction formula Φ over signature Σ is *satisfiable* in a Σ-theory T, or is T-satisfiable, when there is an element of the set T that satisfies Φ. For example, restriction formula (4.2) is satisfiable in any Σ_1-theory that contains interpretation $\mathscr{I}_2$. On the other hand, restriction formula (4.2) is not satisfiable in Σ_1-theory $\{\mathscr{I}_1\}$.

To conclude the introduction to the concept of Σ-theory: from the semantical perspective, it is a collection of Σ-interpretations; from the syntactic perspective, the theory is the collection of restriction formulas satisfied by these models.

4.1 Uniform theories and link to generalized constraint satisfaction processing

The presented definition of a theory (Definition 8) places no restrictions on the domains, r-denotations, or f-denotations being identical across the interpretations defining a theory. In practice, such restrictions are very common in SMT. We now define "uniform" theories that follow these typical restrictions. We will then show how restriction formulas interpreted over uniform theories can practically be seen as GCSPs.

Definition 9

For a signature Σ, we call a Σ-theory T *uniform* over lexicon $\mathscr{L} = ([\Sigma, D], \rho, \phi)$ when

(1) all interpretations in T are of the form (D, v, ρ, ϕ) (note how valuation v is the only not fixed element in the interpretations), and
(2) for every possible $[\Sigma_{|v}, D]$ valuation v, there is an interpretation (D, v, ρ, ϕ) in T.

Example 6

To illustrate a concept of a uniform theory, a table below defines sample domain D_2, valuations v_3 and v_4, and r-denotation ρ_4.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

D_2	$\{1, 2\}$
v_3	$[\Sigma_{1\|v}, D_2]$ valuation, where $s^{v_3} = 1$ and $r^{v_3} = 2$
v_4	$[\Sigma_{1\|v}, D_2]$ valuation, where $s^{v_4} = 2$ and $r^{v_4} = 2$
ρ_4	$[\Sigma_{1\|r}, D_2]$ r-denotation, where $E^{\rho_4} = \{\langle 2\rangle\}, Q^{\rho_4} = \{\langle 1, 1\rangle, \langle 2, 2\rangle\}$

We also note that valuations v_1 and v_2 from Table 1 can be seen not only as $[\Sigma_{1|v}, D_1]$ valuations, but also as $[\Sigma_{1|v}, D_2]$ valuations.

The set

$$\{(D_2, v_1, \rho_4), (D_2, v_2, \rho_4), (D_2, v_3, \rho_4), (D_2, v_4, \rho_4)\} \tag{4.3}$$

of Σ_1 interpretations is an example of a uniform theory over lexicon $([\Sigma_1, D_2], \rho_4)$. The set

$$\{(D_2, v_1, \rho_4), (D_2, v_2, \rho_4), (D_2, v_3, \rho_4), (D_1, v_4, \rho_4)\}$$

of Σ_1 interpretations is an example of a non-uniform theory. Indeed, the condition 1 of Definition 9 does not hold for this theory: the last interpretation refers to a different domain than the others. Also, recall interpretations $\mathscr{I}_1$ and $\mathscr{I}_2$ given in (4.1). Neither Σ_1-theory $\{\mathscr{I}_1\}$ nor $\{\mathscr{I}_1, \mathscr{I}_2\}$ is uniform over lexicon $([\Sigma_1, D_1], \rho_1)$. In this case, the condition 2 of Definition 9 does not hold.

It is easy to see that for uniform theories we can identify their interpretations of the form (D, v, ρ, ϕ) with their second element – valuation v. Indeed, the other three elements are fixed by the lexicon over which the uniform theory is defined. In the following, we will sometimes use this convention. For example, we may refer to interpretation (D_2, v_1, ρ_4) of uniform theory (4.3) as v_1.

For uniform Σ-theory T over lexicon $([\Sigma, D], \rho, \phi)$, we can extend the syntax of restriction formulas by saying that a *restriction formula* is defined over c-vocabulary $[\Sigma, D]$ as a finite set of constraint literals over $[\Sigma, D]$ (earlier we considered constraint literals over $[\Sigma, \emptyset]$). The earlier definition of semantics is still applicable.

We now present a theorem that makes the connection between GCSPs over a lexicon $\mathscr{L}$ and restriction formulas interpreted using the uniform theory T over the same lexicon apparent. As a result, the question whether a given GCSP over $\mathscr{L}$ has a solution translates into the question whether the set of constraint literals of a GCSP forming a restriction formula is T-satisfiable. Furthermore, any solution to such GCSP is also an interpretation in T that satisfies the respective restriction formula, and the other way around.

Theorem 5

For a lexicon $\mathscr{L} = ([\Sigma, D], \rho, \phi)$, a set Φ of constraint literals over c-vocabulary $[\Sigma, D]$, and a uniform Σ-theory T over lexicon $\mathscr{L}$ the following holds:

(1) for any $[\Sigma_{|v}, D]$ valuation v, there is an interpretation v in T,
(2) $[\Sigma_{|v}, D]$ valuation v is a solution to GCSP Φ over lexicon $\mathscr{L}$ if and only if interpretation v in T satisfies restriction formula Φ,
(3) GCSP Φ over lexicon $\mathscr{L}$ has a solution if and only if restriction formula Φ is T-satisfiable.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Proof
Statement 1 trivially follows from the condition 2 of the definition of uniform theories.

Proof of Statement 2. By Statement 1, interpretation v is in T. By definition of a solution to GCSP, $[\Sigma_{|v}, D]$ valuation v is a solution to GCSP Φ over lexicon $\mathscr{L}$ if and only if v is a solution to every constraint in Φ. In other words, v satisfies every constraint literal in Φ with respect to ρ and ϕ. By definition of interpretations satisfying formulas, the previous statement holds if and only if interpretation v in T satisfies Φ.

Proof of Statement 3. GCSP Φ over lexicon $\mathscr{L}$ has a solution if and only if there is a $[\Sigma_{|v}, D]$ valuation v that is a solution to GCSP Φ over lexicon $\mathscr{L}$. By statements 1 and 2, the previous statement holds if and only if there is an interpretation v in T that satisfies Φ and consequently Φ is T-satisfiable. □

Such commonly used theories in SMT as *linear real arithmetic*, *linear integer arithmetic*, and *integer difference logic* are uniform.[3] To be more precise, recall notions of numeric, integer lexicons, and (integer) linear constraints are presented in Section 2.2.1. Given these notions:

- Linear real arithmetic is an example of a uniform theory over a numeric lexicon. This arithmetic poses syntactic conditions on restriction formulas that it interprets. Namely, literals in these restriction formulas must correspond to linear constraints.
- Similarly, linear integer arithmetic and integer difference logic are examples of uniform theories over integer lexicons. Literals in restriction formulas in these arithmetics must correspond to integer linear constraints. Furthermore, the difference logic is a special case of linear integer arithmetic posing yet additional syntactic restrictions (Nieuwenhuis and Oliveras 2005).

From Theorem 5, it follows that restriction formulas in linear real arithmetic can be seen as linear CSPs (and the other way around). Similar relation holds between restriction formulas in linear integer arithmetic and integer linear CSPs.

5 SMT formulas and ASPT programs

First, this section introduces SMT formulas that merge the concepts of propositional formulas and Σ-theories. Second, it introduces ASPT programs that merge the concepts of logic programs and Σ-theories. It turns out that if considered Σ-theories are uniform, then formalisms of constraint formulas and SMT formulas coincide. A similar relation holds of constraint answer set programs and ASPT programs. This link is discussed in the next section.

As in Section 3.1, we understand σ_r and σ_i as two disjoint vocabularies and refer to their elements as regular and irregular.

[3] For instance, the SMT solver CVC4 (Barrett *et al.* 2011) supports all three mentioned arithmetics.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Definition 10
An *SMT formula* P over vocabulary $\sigma = \sigma_r \cup \sigma_i$ is a triple $\langle F, T, \mu \rangle$, where

- F is a propositional formula over σ,
- T is a Σ-theory, and
- μ is an injective function from irregular atoms σ_i to constraint literals over c-vocabulary $[\Sigma, \emptyset]$.

For an SMT formula $\langle F, T, \mu \rangle$ over σ, a set $X \subseteq At(F)$ is its *model* if

(1) X is a model of F, and
(2) the following restriction formula

$$\{\mu(a) | a \in X \cap \sigma_i\} \cup \{\neg\mu(a) | a \in (At(F) \cap \sigma_i) \setminus X\}$$

is satisfiable in Σ-theory T.

In the literature on SMT, a more sophisticated syntax than SMT formulas provide is typically discussed. Yet, SMT solvers often rely on the so-called propositional abstractions of predicate logic formulas (Barrett *et al.* 2008, Section 26.2.1.4) or (Barrett and Tinelli 2014, Section 1.1), which, in their most commonly used case, coincide with SMT formulas discussed here.

We now define the concept of logic programs modulo theories.

Definition 11
A *logic program modulo theories* (or *ASPT program*) P over vocabulary $\sigma = \sigma_r \cup \sigma_i$ is a triple $\langle \Pi, T, \mu \rangle$, where

- Π is a logic program over σ such that $hd(\Pi) \cap \sigma_i = \emptyset$,
- T is a Σ-theory, and
- μ is an injective function from irregular atoms σ_i to constraint literals over c-vocabulary $[\Sigma, \emptyset]$.

For an ASPT program $\langle \Pi, T, \mu \rangle$ over σ, a set $X \subseteq At(\Pi)$ is its *answer set* if

(1) X is an input answer set of Π relative to σ_i, and
(2) the following restriction formula

$$\{\mu(a) | a \in X \cap \sigma_i\} \cup \{\neg\mu(a) | a \in (At(\Pi) \cap \sigma_i) \setminus X\}$$

is satisfiable in Σ-theory T.

In the case of uniform theories, we can extend the definition of SMT formula given a constraint Σ-theory T over lexicon $([\Sigma, D], \rho, \phi)$ as follows: an *SMT formula* P over vocabulary $\sigma = \sigma_r \cup \sigma_i$ is a triple $\langle F, T, \mu \rangle$, where F is a propositional formula over σ, T is a Σ-theory, and μ is an injective function from irregular atoms σ_i to constraint literals over c-vocabulary $[\Sigma, D]$. Note how the only difference in this definition is that function μ refers to domain D of lexicon in place of an empty set. The definition of ASPT program can be extended in the same style. For the case of uniform theories, we will assume the definitions of SMT formulas and ASPT programs as stated in this paragraph.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

6 SMT formulas versus constraint Formulas and ASPT versus CAS programs

The framework of uniform theories brings us to a straightforward relation between SMT formulas over uniform theories and constraint formulas and between CAS programs and ASPT programs. We now formalize these statements.

Let $\mathscr{L}$ denote a lexicon $([\Sigma, D], \rho, \phi)$. By $\mathscr{B}_{\mathscr{L}}$ we denote the set of all constraints over $\mathscr{L}$. By $T_{\mathscr{L}}$ we denote the uniform Σ-theory over $\mathscr{L}$.

Theorem 6
For a lexicon $\mathscr{L} = ([\Sigma, D], \rho, \phi)$, a vocabulary $\sigma = \sigma_r \cup \sigma_i$, and a set X of atoms over σ, set X is a model of SMT formula $\langle F, T_{\mathscr{L}}, \mu\rangle$ over σ if and only if X is a model of a constraint formula $\langle F, \mathscr{B}_{\mathscr{L}}, \mu\rangle$ over σ (where μ is identified with the function from irregular atoms to constraints over $\mathscr{L}$ in a trivial way.)

Proof
Let X be a subset of $At(F)$. Set X is a model of an SMT formula $\langle F, T_{\mathscr{L}}, \mu\rangle$ over σ if and only if

(1) X is a model of F, and
(2) the following restriction formula

$$\{\mu(a) | a \in X \cap \sigma_i\} \cup \{\neg\mu(a) | a \in (At(F) \cap \sigma_i) \setminus X\} \tag{6.1}$$

is satisfiable in Σ-theory $T_{\mathscr{L}}$.

By the definition of a model of a constraint formula, to conclude the proof, we only have to illustrate that condition 2 holds if and only if the GCSP (6.1) over $\mathscr{L}$ has a solution. By Statement 3 of Theorem 5 we conclude that GSCP (6.1) has a solution if and only if restriction formula (6.1) is satisfiable in Σ-theory $T_{\mathscr{L}}$ (which is the case if and only if condition 2 holds). □

This theorem illustrates that for uniform theories the languages of SMT formulas and constraint formulas practically coincide. In other words, constraint formulas is a special case of SMT formulas that are defined over uniform theories. It is obvious that a similar relation between CAS and ASPT programs holds.

Theorem 7
For a lexicon $\mathscr{L} = ([\Sigma, D], \rho, \phi)$, a vocabulary $\sigma = \sigma_r \cup \sigma_i$, and a set X of atoms over σ, set X is an answer set of ASPT program $\langle \Pi, T_{\mathscr{L}}, \mu\rangle$ over σ if and only if X is an answer set of a CAS program $\langle \Pi, \mathscr{B}_{\mathscr{L}}, \mu\rangle$ over σ (where μ is identified with the function from irregular atoms to constraints over $\mathscr{L}$ in a trivial way.)

The proof for this theorem follows the lines of proof of Theorem 6.

Example 7
Recall Examples 3 and 5. Let $T_{\mathscr{L}_3}$ denote the uniform theory over integer lexicon $\mathscr{L}_3$ defined in Example 3.

By Theorem 7, CAS program $P_1 = \langle \Pi_1, \mathscr{B}_{\mathscr{L}_3}, \gamma_1\rangle$ from Example 3 is essentially the same entity as ASPT program $\langle \Pi_1, T_{\mathscr{L}_3}, \gamma_1\rangle$. These programs have the same answer sets.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Similarly, by Theorem 6, constraint formula

$$\mathscr{F}_1 = \langle IComp(\Pi_1, \{|x < 12|, |x \geqslant 12|, |x < 0|, |x > 23|\}), \mathscr{B}_{\mathscr{L}_3}, \gamma_1 \rangle$$

from Example 5 is essentially the same entity as SMT formula

$$\langle IComp(\Pi_1, \{|x < 12|, |x \geqslant 12|, |x < 0|, |x > 23|\}), T_{\mathscr{L}_3}, \gamma_1 \rangle.$$

We call any SMT formula $\langle F, T, \mu \rangle$ over $\sigma_r \cup \sigma_i$

- an *SMT(IL) formula* if T is the uniform theory over an integer lexicon and μ maps irregular atoms σ_i into integer linear constraints;
- an *SMT(DL) formula* if T is the uniform theory over an integer lexicon and μ maps irregular atoms σ_i into difference logic constraints;
- an *SMT(L) formula* if T is the uniform theory over a numeric lexicon and μ maps irregular atoms σ_i into linear constraints.

In the same style, we can define ASPT(IL), ASPT(DL), and ASPT(L) programs.

From Theorem 7, CAS programs of the form $\langle \Pi, \mathscr{B}_{\mathscr{L}}, \gamma \rangle$, where $\mathscr{L}$ is a numeric lexicon and $\mathscr{B}_{\mathscr{L}}$ is the set of all linear constraints over $\mathscr{L}$, are essentially the same objects as ASPT(L) programs. Similarly, it follows that CAS programs of the form $\langle \Pi, \mathscr{B}_{\mathscr{L}}, \gamma \rangle$, where $\mathscr{L}$ is an integer lexicon and $\mathscr{B}_{\mathscr{L}}$ is the set of all integer linear constraints over $\mathscr{L}$, are essentially the same objects as ASPT(IL) programs.

Obviously, Theorems 4 and 6 pave the way for using the SMT systems that solve SMT(IL) and SMT(L) problems as is for solving tight ASPT(IL) and ASPT(L) programs, respectively. It is sufficient to compute the input completion of the program relative to irregular atoms. Susman and Lierler (2016) utilized this method in implementing SMT-based solver for tight programs called EZSMT. A similar observation has been used in work by Lee and Meng (2013) and Janhunen *et al.* (2011). Furthermore, Janhunen *et al.* propose a translation of ASPT(DL) programs into SMT(DL) formulas. System DINGO utilizes this translation by invoking SMT solver z3 for finding models for ASPT(DL) programs.

6.1 Outlook on constraint answer set solvers

We now relate various constraint answer set solvers by specifying which ASPT languages these solvers support. We also provide a brief overview of a variety of solving techniques that they use.

Table 3 presents the landscape of current constraint answer set solvers. The star $*$ annotating language ASPT(IL) denotes that the solver supporting this language requires the specification of finite ranges for its variables (since finite-domain constraint solvers are used as underlying solving technology). Symbol † that annotates the languages supported by system MINGO represents the fact that this solver supports lexicons that are not covered by our framework. Its variables within the same program can be of two different kind: either integer or real.

At a high-level abstraction, one may summarize the architectures of the CLINGCON and EZCSP solvers as *ASP-based solvers plus theory solver*. Given a CAS program $\langle \Pi, \mathscr{B}, \gamma \rangle$, both CLINGCON and EZCSP first use an answer set solver to partially compute

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Table 3. *Solvers categorization*

Solver	Language
CLINGCON (Gebser *et al.* 2009)	ASPT(IL)*
MINGO (Liu *et al.* 2012)	ASPT(IL)†
	ASPT(L)†
DINGO (Janhunen *et al.* 2011)	ASPT(DL)
EZCSP (Balduccini 2009)	ASPT(IL)*
	ASPT(IL)
	ASPT(L)
EZSMT (Susman and Lierler 2016)	ASPT(IL)
	ASPT(L)

an input answer set of Π. Second, they contact a theory solver to verify whether respective CSP has a solution. In case of CLINGCON, finite-domain constraint solver GECODE is used as a theory solver. System EZCSP uses constraint logic programming tools such as BPROLOG (Zhou 2012), SICSTUS PROLOG (Carlsson and Fruehwirth 2014), and SWI PROLOG (Wielemaker *et al.* 2012). These tools provide EZCSP with the ability to work with three different kinds of constraints: finite-domain integer, integer linear, and linear constraints. To process ASPT(IL) and ASPT(L) programs, the solver MINGO translates these programs into mixed integer programming expressions and then uses the solver CPLEX (IBM 2009) to solve these formulas. To process ASPT(DL) programs DINGO translates these programs into SMT(DL) formulas and applies the SMT solver Z3 (De Moura and Bjørner 2008) to find their models. The EZSMT solver is only capable of processing tight programs. It computes classified input completion for such CAS programs, encodes resulting SMT formula in SMT-LIB language (Barrett *et al.* 2015) and is then able to invoke any SMT solver that supports SMT-LIB (for instance, SMT solvers Z3 (De Moura and Bjørner 2008) or CVC4 (Barrett *et al.* 2011)) to compute its models.

The diversity of solving approaches used in CASP paradigms suggests that solutions of the kind are available for SMT technology. Typical SMT architecture is in a style of systems CLINGCON and EZCSP. At a high-level abstraction, one may summarize common architectures of SMT solvers as satisfiability-based solvers augmented with theory solvers. Theory solvers are usually implemented within an SMT solver and are as such custom solutions. The fact that CLINGON and EZCSP use tools available from the constraint programming community suggests that these tools could be of use in SMT community also. The solution exhibited by system MINGO, where mixed integer programming is used for solving ASPT(L) and ASPT(IL) programs, hints that a similar strategy can be implemented for solving SMT(L) formulas. These ideas have been explored by King *et al.* (2014).

7 SMT solvers for non-tight programs via level rankings

As mentioned in section on preliminaries, Niemelä (2008) characterized answer sets of normal logic programs in terms of level rankings. He then developed a mapping from such programs to SMT(DL) formalism. Mappings of the kind were exploited

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

in the design of solvers DINGO (Janhunen *et al.* 2011) and MINGO (Liu *et al.* 2012). In this section, we devise a similar mapping and show how it provides means for using SMT solvers for processing non-tight ASPT(IL) programs.

We start by defining level ranking relative to an input vocabulary and then lifting the result of Theorem 1 to the notion of input answer set.

Definition 12
A function $\mathrm{lr} : X \setminus \iota \to \mathbb{N}$ is a *level ranking* of X for Π relative to vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$, when for each atom a in $X \setminus \iota$ the following condition hold: there is B in $Bodies(\Pi, a)$ such that X satisfies B and for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$.

Theorem 8
For a program Π over vocabulary σ, vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$, and a set X of atoms over σ that is a model of input completion $IComp(\Pi, \iota)$, X is an input answer set of Π relative to ι if and only if there is a level ranking of X for Π relative to ι.

Proof
Consider a set X of atoms over σ that is a model of $IComp(\Pi, \iota)$. By Lemma 1, it follows that X is a model of

$$Comp(\Pi \cup (X \cap \iota)). \tag{7.1}$$

Left-to-right: Assume set X is an input answer set of Π relative to ι. By the definition of an input answer set, X is an answer set of the program

$$\Pi \cup (X \cap \iota). \tag{7.2}$$

By Theorem 1, it follows that there is a level ranking of X for program (7.2). Thus, by Definition 1, there is a function $\mathrm{lr}: X \to \mathbb{N}$ such that for each $a \in X$, there is B in $Bodies(\Pi \cup (X \cap \iota), a)$ such that X satisfies B and for every $b \in B^+$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$. It is easy to see that for each atom a in $X \setminus \iota$,

$$Bodies(\Pi \cup (X \cap \iota), a) = Bodies(\Pi, a). \tag{7.3}$$

Thus, for each atom a in $X \setminus \iota$ there is B in $Bodies(\Pi, a)$ such that X satisfies B and for every $b \in B^+$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$, and hence for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$. By Definition 12, lr (seen as a function from $X \setminus \iota$ to $\mathbb{N}$) is a level ranking of X for Π relative to ι.

Right-to-left: Assume there is a level ranking lr of X for Π relative to ι. By Definition 12, for each atom a in $X \setminus \iota$, there is B in $Bodies(\Pi, a)$ such that X satisfies B and for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$.

We now construct function $\mathrm{lr}' : X \to \mathbb{N}$ based of lr. We then illustrate that lr' is a level ranking of X for program (7.2).

For every atom a in $X \cap \iota$, $\mathrm{lr}'(a) = 0$. For every atom a in $X \setminus \iota$, $\mathrm{lr}'(a) = \mathrm{lr}(a) + 1$. It is easy to see that

$$(X \cap \iota) \ \cup \ (X \setminus \iota) = X \text{ as well as } (X \cap \iota) \ \cap \ (X \setminus \iota) = \emptyset.$$

Thus, lr' is a function from X to $\mathbb{N}$. Consider two cases.

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Case 1. Atom a in $X \cap \iota$. Since $hd(\Pi) \cap \iota = \emptyset$, a does not appear in any head in Π (i.e., $a \notin hd(\Pi)$), and thus it only appears as a fact in $\Pi \cup (X \cap \iota)$. Trivially, level ranking condition for this atom is satisfied.

Case 2. Atom a in $X \setminus \iota$. It is easy to see that for such an atom equality (7.3) holds. We are given that there is B in $Bodies(\Pi, a)$ such that X satisfies B and for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$. From lr' construction, it follows that for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}'(a) - 1 \geqslant \mathrm{lr}'(b)$. On the other hand, for every atom b in B^+ that it is not in $B^+ \setminus \iota$, such b is in $X \cap \iota$. By lr' construction, $\mathrm{lr}'(b) = 0$ and $\mathrm{lr}'(a) = \mathrm{lr}(a) + 1$. Since lr is a mapping to natural numbers we derive that $\mathrm{lr}(a) \geqslant 0$ and hence $\mathrm{lr}'(a) \geqslant 1$. Thus, $\mathrm{lr}'(a) - 1 \geqslant \mathrm{lr}'(b)$ holds. This concludes our illustration that for every atom $a \in X$ the level ranking condition holds given function lr'.

By Theorem 1 we conclude that X is an answer set of program (7.2). Consequently, X is an input answer set of Π relative to ι. □

We now present a mapping from a logic program to SMT(IL) formula inspired by the mapping introduced by Niemelä (2008). The mapping $T(\Pi, \iota)$ consists of input-completion $IComp(\Pi, \iota)$ and ranking-formula $R(\Pi, \iota)$ defined below. Sometimes, we refer to the set of formulas as a formula, where we understand such a formula as a conjunction of the members of its set.

In defining $R(\Pi, \iota)$, we will use the convention discussed earlier so that vertical bars mark the irregular atoms that have intuitive mappings into respective integer linear constraints. For instance, expression $|lr_b - 1 \geqslant lr_a|$ introduces an irregular atom that is intuitively mapped into integer linear constraint $lr_b - 1 \geqslant lr_a$, where lr_a and lr_b are variables over integers. Expression $R(\Pi, \iota)$ is a conjunction of formulas constructed as follows: for each atom $a \in \sigma \setminus \iota$

$$a \rightarrow \bigvee_{a \leftarrow B \in \Pi \text{ and } B^+ \setminus \iota \neq \emptyset} (B \wedge \bigwedge_{b \in B^+ \setminus \iota} |lr_a - 1 \geqslant lr_b|) \vee \bigvee_{a \leftarrow B \in \Pi \text{ and } B^+ \setminus \iota = \emptyset} B \tag{7.4}$$

By $\sigma_i^{R(\Pi,\iota)}$ we denote the set of all irregular atoms of the form $|lr_a - 1 \geqslant lr_b|$ in $R(\Pi, \iota)$. By $\Sigma^{R(\Pi,\iota)}$ we denote the set of all variables over integers occurring within $\sigma_i^{R(\Pi,\iota)}$. By $\gamma^{R(\Pi,\iota)}$ we denote a mapping from irregular atoms in $\sigma_i^{R(\Pi,\iota)}$ to integer linear constraints in accordance to their intuitive meanings. By $\mathscr{B}_{R(\Pi,\iota)}$ we denote the set of all integer linear constraints over integer lexicon defined over the signature $\Sigma^{R(\Pi,\iota)}$.

Theorem 9

For a program Π over vocabulary σ, vocabulary $\iota \subseteq \sigma$ so that $hd(\Pi) \cap \iota = \emptyset$, and a set X of atoms over σ, X is an input answer set of Π relative to ι if and only if there is a model $X \cup X_i$ of a constraint formula (or SMT(IL) formula)

$$\langle IComp(\Pi, \iota) \wedge R(\Pi, \iota), \mathscr{B}_{R(\Pi,\iota)}, \gamma^{R(\Pi,\iota)} \rangle \tag{7.5}$$

over $\sigma \cup \sigma_i^{R(\Pi,\iota)}$ so that $X_i \subseteq \sigma_i^{R(\Pi,\iota)}$.

Proof

Left-to-right: Assume X is an input answer set of Π relative to ι. By Theorem 3, it follows that X satisfies $IComp(\Pi, \iota)$. By Theorem 8, there is a level ranking lr of

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

X relative to ι. Consider such lr. By the level ranking definition, for each atom a in $X \setminus \iota$ the following condition holds, there is B in $Bodies(\Pi, a)$ such that X satisfies B and for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$. We now construct set X_i of atoms over $\sigma_i^{R(\Pi,\iota)}$ as follows: atom $|lr_a - 1 \geqslant lr_b|$ from $\sigma_i^{R(\Pi,\iota)}$ is in X_i if and only if condition $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$ holds.

It follows immediately from the construction of X_i that the GCSP which corresponds to $X \cup X_i$ (in accordance with condition 2 in the definition of a model of a constraint formula) has a solution. Indeed, consider a valuation v that assigns values to variables of the form lr_a in $\Sigma^{R(\Pi,\iota)}$ based on level ranking function lr. In particular, $\mathrm{lr}_a^v = \mathrm{lr}(a)$.

We are now left to illustrate that $X \cup X_i$ is a model of $IComp(\Pi, \iota) \wedge R(\Pi, \iota)$. Since X is a model of $IComp(\Pi, \iota)$, we are only left to show that $X \cup X_i$ is a model of $R(\Pi, \iota)$ or, in other words, that $X \cup X_i$ satisfies (7.4) for every atom $a \in \sigma \setminus \iota$.

Consider any atom $a \in \sigma \setminus \iota$.

Case 1. $a \notin X$. Obviously, $a \notin X_i$ since $X_i \cap \sigma = \emptyset$. Then, $X \cup X_i$ trivially satisfies (7.4).

Case 2. $a \in X \setminus \iota$. Since lr is a level ranking of X relative to ι it follows that there is $B \in Bodies(\Pi, a)$ so that $X \models B$ and for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$. Take such B.

Case 2.1. $B^+ \setminus \iota = \emptyset$. Then, X satisfies (7.4) due to

$$\bigvee_{a \leftarrow B \in \Pi \text{ and } B^+ \setminus \iota = \emptyset} B$$

term in the right-hand side of the implication (7.4) and the fact that X satisfies considered B. Consequently, $X \cup X_i$ satisfies (7.4).

Case 2.2. $B^+ \setminus \iota \neq \emptyset$. From the construction of X_i, it follows that for every $b \in B^+ \setminus \iota$ there is an irregular atom of the form $|\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)|$ in X_i. Then, $X \cup X_i$ satisfies (7.4) due to

$$\bigvee_{a \leftarrow B \in \Pi \text{ and } B^+ \setminus \iota \neq \emptyset} (B \wedge \bigwedge_{b \in B^+ \setminus \iota} |lr_a - 1 \geqslant lr_b|)$$

term in the right-hand side of the implication (7.4).

Right-to-left: Let $X \cup X_i$ be a model of constraint formula (7.5). It immediately follows that X is a model of $IComp(\Pi, \iota)$. We now show that one can construct a level ranking lr of X for Π relative to ι using X_i. Indeed, consider a GCSP that corresponds to $X \cup X_i$ (in accordance with the condition 2 in the definition of a model of a constraint formula). Since $X \cup X_i$ is a model of (7.5) that GCSP has a solution, we use such a solution v to construct level ranking lr for each atom a in $X \setminus \iota$ as follows:

$$\mathrm{lr}(a) = \begin{cases} lr_a^v & \text{if } lr_a \in \Sigma^{R(\Pi,\iota)} \\ 0 & \text{otherwise} \end{cases}$$

To verify that lr is indeed a level ranking of X for Π relative to ι using X_i we have to illustrate that for every atom $a \in X \setminus \iota$ there is $B \in Bodies(\Pi, a)$ such that X satisfies B and for every $b \in B^+ \setminus \iota$ it holds that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$. Consider any

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

atom $a \in X \setminus \iota$. We are given that $X \cup X_i$ is a model of $R(\Pi, \iota)$. Thus, the right-hand side of the implication (7.4) is satisfied for chosen atom a. It follows that there is $B \in Bodies(\Pi, a)$ such that $X \models B$ and for every $b \in B^+ \setminus \iota$, X_i contains the following irregular atom $|lr_a - 1 \geqslant lr_b|$. From the fact that v is a solution to the GCSP that includes an integer linear constraint $lr_a - 1 \geqslant lr_b$ it follows that inequality $lr_a^v - 1 \geqslant lr_b^v$ holds. By lr construction we conclude that $\mathrm{lr}(a) - 1 \geqslant \mathrm{lr}(b)$.

By Theorem 8, X is an input answer set of Π relative to ι. □

Theorem 10

Let $\mathscr{L}$ be an integer lexicon and $\mathscr{B}_{\mathscr{L}}$ be the set of all integer linear constraints over $\mathscr{L}$. For a CASP Program (or, an ASPT(IL) program) $P = \langle \Pi, \mathscr{B}_{\mathscr{L}}, \gamma \rangle$ over vocabulary $\sigma = \sigma_r \cup \sigma_i$, and a set X of atoms from σ, X is an answer set of P if and only if there is a model $X \cup X_i$ of a constraint formula (or SMT(IL formula))

$$\langle IComp(\Pi, \sigma_i) \wedge R(\Pi, \sigma_i), \mathscr{B}_{\mathscr{L}} \cup \mathscr{B}_{R(\Pi,\iota)}, \gamma' \rangle$$

over $\sigma_r \cup \sigma_i \cup \sigma_i^{R(\Pi,\iota)}$ so that X_i is the set of atoms over $\sigma_i^{R(\Pi,\iota)}$ and γ' is such that it coincides with γ on atoms in σ_i and with $\gamma^{R(\Pi,\iota)}$ on atoms from $\sigma_i^{R(\Pi,\iota)}$.

Proof of Theorem 10 follows the lines of proof for Theorem 9.

Theorem 10 paves the way for using SMT solvers for computing answer sets for arbitrary ASPT(IL) programs. Niemelä (2008) introduced the notions of strong level ranking and also illustrated how strongly connected components of a dependency graph of a normal program can be used to enhance the transformation from a normal program to an SMT(DL) formula. Similar ideas could be used for enhancing the proposed translation from ASPT(IL) to SMT(IL) formalism. Such enhancements are of essence when implementation of SMT-based solver for nontight ASPT(IL) programs is considered. Implementing such enhancements is the direction of future work.

8 Conclusions

In this paper, we unified the terminology stemming from the fields of CASP and SMT solving. This unification helped us identify the special class of uniform theories widely used in SMT practice. Given such theories, CASP and SMT solving share more in common than meets the eye. Based on this unification, we open the doors for writing programs in the CASP formalism, while allowing SMT solving technologies to be utilized. In these settings, CASP can be seen as a possible general-purpose declarative programming front-end for SMT technology. In the future, we would like to investigate a similar link to a related formalism of HEX-programs (Eiter *et al.* 2012). Overall, we expect this work to be a strong building block that will bolster the cross-fertilization between three different, even if related, automated reasoning communities: CASP, constraint (satisfaction processing) programming, and SMT.

References

Balduccini, M. 2009. Representing constraint satisfaction problems in answer set programming. In *Proc. of ICLP Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*. URL: `https://www.mat.unical.it/ASPOCP09/`

https://doi.org/10.1017/S1471068417000114 Published online by Cambridge University Press

Balduccini, M. 2013. ASP with non-Herbrand partial functions: A language and system for practical use. *Theory and Practice of Logic Programming 13*, 547–561.

Balduccini, M. and Lierler, Y. 2017. Constraint answer set solver EZCSP and why integration schemas matter. *Theory and Practice of Logic Programming*, in this issue.

Barrett, C., Conway, C. L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A. and Tinelli, C. 2011. CVC4. In *Proc. of 23rd International Conference on Computer Aided Verification (CAV'11)*, Lecture Notes in Computer Science, vol. 6806. Springer International Publishing.

Barrett, C., Fontaine, P. and Tinelli, C. 2015. The SMT-LIB standard: Version 2.5. Technical Report, Department of Computer Science, The University of Iowa. URL: `www.SMT-LIB.org`

Barrett, C., Sebastiani, R., Seshia, S. and Tinelli, C. 2008. Satisfiability modulo theories. In *Handbook of Satisfiability*, A. Biere, M. Heule, H. van Maaren and T. Walsch, Eds. IOS Press, 737–797.

Barrett, C. and Tinelli, C. 2014. Satisfiability modulo theories. In *Handbook of Model Checking*, E. Clarke, T. Henzinger and H. Veith, Eds. Springer International Publishing.

Bartholomew, M. and Lee, J. 2012. Stable models of formulas with intensional functions. In *Proc. of International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Bomanson, J., Gebser, M., Janhunen, T., Kaufmann, B. and Schaub, T. 2015. Answer set programming modulo acyclicity. In *Proc. of 13th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, Lexington, KY, USA, F. Calimeri, G. Ianni and M. Truszczynski, Eds. Springer International Publishing, Cham, 143–150.

Calimeri, F., Cozza, S., Ianni, G. and Leone, N. 2008. Computable functions in ASP: Theory and implementation. In *Proc. of International Conference on Logic Programming (ICLP)*, 407–424.

Carlsson, M. and Fruehwirth, T. 2014. *SICStus PROLOG User's Manual 4.3*. Books On Demand - Proquest.

Clark, K. 1978. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York, 293–322.

De Moura, L. and Bjørner, N. 2008. Z3: An efficient SMT solver. In *Proc. of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 337–340.

Denecker, M. and Vennekens, J. 2007. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In *Proc. of 9th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR)*, Tempe, AZ, USA, C. Baral, G. Brewka and J. S. Schlipf, Eds. Lecture Notes in Computer Science, vol. 4483. Springer International Publishing, 84–96.

Eiter, T., Fink, M., Krennwallner, T. and Redl, C. 2012. Conflict-driven ASP solving with external sources. *Theory and Practice of Logic Programming 12,* 4–5, 659–679.

Elkabani, I., Pontelli, E. and Son, T. C. 2004. Smodels with CLP and its applications: A simple and effective approach to aggregates in ASP. In *Proc. of International Conference on Logic Programming (ICLP)*, 73–89.

Erdem, E. and Lifschitz, V. 2001. Fages' theorem for programs with nested expressions. In *Proc. of International Conference on Logic Programming (ICLP)*, 242–254.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science 1*, 51–60.

Ferraris, P. and Lifschitz, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5*, 45–74.

Gebser, M., Janhunen, T. and Rintanen, J. 2014. SAT modulo graphs: Acyclicity. In *Proc. of 14th European Conference Logics in Artificial Intelligence (JELIA), Funchal, Madeira, Portugal*, E. Fermé and J. Leite, Eds. Springer International Publishing, Cham, 137–151.

Gebser, M., Ostrowski, M. and Schaub, T. 2009. Constraint answer set solving. In *Proc. of 25th International Conference on Logic Programming (ICLP)*. Springer International Publishing, 235–249.

Gebser, M., Schaub, T. and Thiele, S. 2007. Gringo: A new grounder for answer set programming. In *Proc. of 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, 266–271.

Gelfond, M. and Przymusinska, H. 1996. Towards a theory of elaboration tolerance: Logic programming approach. *International Journal of Software Engineering and Knowledge Engineering 6,* 1, 89–112.

IBM 2009. *IBM ILOG AMPL Version 12.1 User's Guide*. IBM. URL: `http://www.ibm.com/software/commerce/optimization/cplex-optimizer/`

Janhunen, T., Liu, G. and Niemela, I. 2011. Tight integration of non-ground answer set programming and satisfiability modulo theories. In *Proc. of 1st Workshop on Grounding and Transformations for Theories with Variables*.

King, T., Barrett, C. and Tinelli, C. 2014. Leveraging linear and mixed integer programming for SMT. In *Proc. of 14th Conference on Formal Methods in Computer-Aided Design (FMCAD'14)*. FMCAD, Austin, TX, 24:139–24:146.

Lee, J. and Meng, Y. 2013. Answer set programming modulo theories and reasoning about continuous changes. In *Proc. of 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*, Beijing, China, August 3–9, 2013.

Lierler, Y. 2014. Relating constraint answer set programming languages and algorithms. *Artificial Intelligence 207C*, 1–22.

Lierler, Y. and Susman, B. 2016. Constraint answer set programming versus satisfiability modulo theories. In *Proc. of 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 1181–1187.

Lierler, Y. and Truszczynski, M. 2011. Transition systems for model generators — a unifying approach. In *Proc. Theory and Practice of Logic Programming, 27th International Conference on Logic Programming (ICLP'11), Special Issue 11, Issue 4–5*.

Lifschitz, V. 2012. Logic programs with intensional functions. In *Proc. of International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Lifschitz, V., Tang, L. R. and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*, 369–389.

Liu, G., Janhunen, T. and Niemela, I. 2012. Answer set programming via mixed integer programming. In *Proc. of 13th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Marriott, K. and Stuckey, P. J. 1998. *Programming with Constraints: An Introduction*. MIT Press.

Mellarkod, V. S., Gelfond, M. and Zhang, Y. 2008. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence 53,* 1, 251–287.

Niemelä, I. 2008. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence 53*, 313–329.

Niemelä, I. and Simons, P. 2000. Extending the Smodels system with cardinality and weight constraints. In *Logic-Based Artificial Intelligence*, J. Minker, Ed. Kluwer, 491–521.

Nieuwenhuis, R. and Oliveras, A. 2005. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Proc. of 17th International Conference on Computer*

Aided Verification (CAV'05), Lecture Notes in Computer Science, vol. 3576. Springer International Publishing.

Oikarinen, E. and Janhunen, T. 2006. Modular equivalence for normal logic programs. In *Proc. of 17th European Conference on Artificial Intelligence (ECAI'06)*, G. Brewka, S. Coradeschi, A. Perini and P. Traverso, Eds. IOS Press, Amsterdam, The Netherlands, 412–416.

Susman, B. and Lierler, Y. 2016. SMT-based constraint answer set solver EZSMT (system description). In *Proc. of 32th International Conference on Logic Programming (ICLP)*. Dagstuhl Publishing, OpenAccess Series in Informatics (OASIcs).

Wielemaker, J., Schrijvers, T., Triska, M. and Lager, T. 2012. SWI-Prolog. *Theory and Practice of Logic Programming 12,* 1–2, 67–96.

Zhou, N. 2012. The language features and architecture of B-Prolog. *Theory and Practice of Logic Programming 12,* 1–2, 189–218.