Constraint Answer Set Programming versus Satisfiability Modulo Theories *

Yuliya Lierler and **Benjamin Susman**
University of Nebraska at Omaha
ylierler@unomaha.edu and bsusman@unomaha.edu

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 make the link between these two areas precise.

Introduction

Constraint answer set programming (CASP) [Mellarkod *et al.*, 2008; Gebser *et al.*, 2009; Balduccini, 2009; Lierler, 2014] is a promising research direction that integrates answer set programming (ASP), a powerful knowledge representation paradigm, with constraint processing. Typical answer set programing tools start their computation with grounding, a process that substitutes variables for passing constants in respective domains. Thus large domains often form an obstacle for classical ASP. CASP enables a mechanism to model constraints over large domains so that they are processed in a non-typical way for ASP 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.

Constraint answer set programming 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.

In this work we attempt 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 are 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 constraint formulas and SMT formulas, and (ii) between CASP and SMT. We are able to then provide a formal description relating a family of distinct constraint answer set programming formalisms.

We believe 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 CASP solvers design as well as SMT solvers design. For example, theoretical results of this work establish the method for using SMT systems for computing answer sets of a broad class of "tight" constraint answer set programs. Similarly, CASP technology can be used to solve certain classes of SMT problems.

The outline of the paper is as follows. We start by reviewing concepts of logic programs, completion, and (input) answer sets. We then present (i) generalized constraint satisfaction problems, (ii) constraint answer set programs, and (iii) constraint formulas. Next we introduce satisfiability modulo theories and respective SMT formulas. We define a class of uniform theories and establish links between CASP and SMT. The paper concludes by relating a family of distinct constraint answer set programming formalisms.

Logic Programs, Completion, and Input Answer Sets

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

$$a \leftarrow b_1, \dots, b_\ell,\ not\ b_{\ell+1}, \dots,\ not\ b_m, \qquad (1)$$

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

$$a \leftarrow B \qquad (2)$$

*We would like to thank Martin Brain for the discussions that lead us to undertake this research. We are also grateful to Vladimir Lifschitz, Cesare Tinelli, and Mirek Truszczynski for the conversations related to the topic of this paper.

where B stands for $b_1, \ldots, b_\ell,\ \mathit{not}\ b_{\ell+1}, \ldots,\ \mathit{not}\ b_m$, and is also called a *body*. We sometimes identify B with the propositional formula $b_1 \wedge \ldots \wedge b_\ell \wedge \neg b_{\ell+1} \wedge \ldots \wedge \neg b_m$ and note that the order of its terms is immaterial. 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 nonempty heads of rules in Π. We refer the reader to the paper by Gelfond and Lifschitz [1988] for details on the definition of an *answer set*.

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 a set of facts $\{a. \mid a \in X\}$. Also, 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 $\sigma \setminus X$.

For a program Π over vocabulary σ, the *completion* of Π [Clark, 1978], denoted by $Comp(\Pi)$, is the set of classical formulas that consists of the implications $B \rightarrow a$ for all rules (2) in Π and the implications

$$a \rightarrow \bigvee_{a \leftarrow B \in \Pi} B \tag{3}$$

for all atoms a in σ.

It is well known that for the large class of logic programs, referred to as "tight" programs, its answer sets coincide with models of its completion, as shown by Fages [1994]. 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 (i) the vertices of G are the atoms occurring in Π, and (ii) for every rule (1) in Π whose head is not $\bot$, G has an edge from atom a to each atom $b_1, \ldots, b_\ell$. A program is called *tight* if its dependency graph is acyclic.

We now introduce a generalization of a concept of an input answer set by Lierler and Truszczynski [2011]. In this work, we consider input answer sets relative to input vocabularies. We then extend the definition of completion so that we can state the result by Fages for the case of input answer sets. These concepts are essential for introducing constraint answer set programs and constraint formulas as they 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 1 *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 $\iota \subseteq \sigma$ when X is an answer set of the program $\Pi \cup ((X \cap \iota) \setminus hd(\Pi))$.*

Definition 2 *For a program Π over vocabulary σ, the* input-completion *of Π relative to vocabulary $\iota \subseteq \sigma$, denoted by $IComp(\Pi, \iota)$, is defined as the set of propositional formulas (formulas in propositional logic) that consists of the implications $B \rightarrow a$ for all rules (2) in Π and the implications (3) for all atoms a occurring in $(\sigma \setminus \iota) \cup hd(\Pi)$.*

Theorem 1 *For a tight program Π over vocabulary σ and vocabulary $\iota \subseteq \sigma$, a set X of atoms from σ is an input answer set of Π relative to ι if and only if X satisfies program's input-completion $IComp(\Pi, \iota)$.*

Constraint Answer Set Programs

We start this section by presenting primitive constraints as defined by Marriott and Stuckey [1998, Section 1.1] using the notation convenient for our purposes. We refer to this concept as a constraint dropping the word "primitive". We use constraints to define a notion of a generalized constraint satisfaction problem that Marriott and Stuckey refer to as "constraint". We then review constraint satisfaction problems as commonly defined in artificial intelligence literature and illustrate that they are special case of generalized constraint satisfaction problems.

Constraints and Generalized Constraint Satisfaction Problem We adopt the following convention: for a function ν and an element x, by x^ν we denote the value that function ν maps x to (in other words, $x^\nu = \nu(x)$). A *domain* is a *nonempty* set of elements (values). A *signature* Σ is a set of *variables*, *function symbols (or f-symbols)*, and *predicate symbols*. Function and predicate 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 function $\nu : V \rightarrow D$ a $[V, D]$ *valuation*. For a set F of f-symbols, we call a total function on F *an* $[F, D]$ *f-denotation*, when it maps an n-ary f-symbol into a function $D^n \rightarrow D$. For a set R of predicate symbols, we call a total function on R *an* $[R, D]$ *r-denotation*, when it maps an n-ary predicate symbol into an n-ary relation on D.

A table below presents definitions of sample domain D_1, valuations ν_1, ν_2, and r-denotations ρ_1 and ρ_2.

D_1	$\{1, 2, 3\}$
ν_1	$[\Sigma_{1\|v}, D_1]$ valuation, where $s^{\nu_1} = r^{\nu_1} = 1$
ν_2	$[\Sigma_{1\|v}, D_1]$ valuation, where $s^{\nu_2} = 2$ and $r^{\nu_2} = 1$
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 *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 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

$$P(t_1, \ldots, t_n), \tag{4}$$

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 (4) or an expression

$$\neg P(t_1, \ldots, t_n), \tag{5}$$

where $P(t_1, \dots, t_n)$ is a constraint atom over $[\Sigma, D]$. For instance, expressions $\neg E(s)$, $\neg E(2)$, and $Q(r, s)$ are constraint literals over $[\Sigma_1, D_1]$.

Let $[\Sigma, D]$ be a c-vocabulary, ν be a $[\Sigma_{|v}, D]$ valuation, ϕ be a $[\Sigma_{|f}, D]$ f-denotation, and ρ be a $[\Sigma_{|r}, D]$ r-denotation. First, we define recursively a value that valuation ν assigns to a term τ over $[\Sigma, D]$ w.r.t. ϕ. We denote this value by $\tau^{\nu,\phi}$. For a term that is a variable x in $\Sigma_{|v}$, $x^{\nu,\phi} = x^{\nu}$. For a term that is a domain element d in D, $d^{\nu,\phi}$ is d itself. For a term τ of the form $f(t_1, \dots, t_n)$, $\tau^{\nu,\phi}$ is defined recursively by the formula $f(t_1, \dots, t_n)^{\nu,\phi} = f^{\phi}(t_1^{\nu,\phi}, \dots, t_n^{\nu,\phi})$. Second, we define what it means for valuation to be a solution of a constraint literal w.r.t. given f- and r-denotations. We say that ν *satisfies (is a solution to)* constraint literal (4) over $[\Sigma, D]$ *w.r.t.* ϕ *and* ρ when $\langle t_1^{\nu,\phi}, \dots, t_n^{\nu,\phi} \rangle \in P^{\rho}$. Let $\mathcal{R}$ be an n-ary relation on D. By $\overline{\mathcal{R}}$ we denote *complement* relation of $\mathcal{R}$ constructed as $D^n \setminus \mathcal{R}$. Valuation ν *satisfies (is a solution to)* constraint literal of the form (5) *w.r.t.* ϕ *and* ρ when $\langle t_1^{\nu,\phi}, \dots, t_n^{\nu,\phi} \rangle \in \overline{P^{\rho}}$. For instance, valuation ν_1 satisfies constraint literal $Q(r, s)$ w.r.t. ρ_1, while valuation ν_2 does not satisfy this constraint literal w.r.t. ρ_2 (when a signature contains no function symbols no reference to f-denotation is necessary in the definitions above).

We are now ready to define constraints, their syntax and semantics. To begin we introduce a *lexicon*, which is a tuple $([\Sigma, D], \rho, \phi)$, where $[\Sigma, D]$ is a c-vocabulary, ρ is $[\Sigma_{|r}, D]$ r-denotation, and ϕ is $[\Sigma_{|f}, D]$ f-denotation. For a lexicon $\mathcal{L} = ([\Sigma, D], \rho, \phi)$, we call any function that is $[\Sigma_{|v}, D]$ valuation, a *valuation over* $\mathcal{L}$. We will omit the last element of the tuple if the signature Σ of the lexicon contains no f-symbols. A *constraint* is defined over lexicon $\mathcal{L} = ([\Sigma, D], \rho, \phi)$. Syntactically, it is a constraint literal over $[\Sigma, D]$ (lexicon $\mathcal{L}$, respectively). Semantically, we say that valuation ν over $\mathcal{L}$ *satisfies (is a solution to)* the constraint c when ν satisfies c w.r.t. ϕ and ρ. For instance, the table below presents definitions of sample lexicons $\mathcal{L}_1$, $\mathcal{L}_2$, and constraints c_1, c_2, c_3, and c_4.

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

Valuation ν_1 is a solution to c_1, c_2, c_3, but not a solution to c_4. Valuation ν_2 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 3 *A* generalized constraint satisfaction problem (GCSP) $\mathcal{C}$ *is a finite set of constraints over a lexicon* $\mathcal{L} = ([\Sigma, D], \rho, \phi)$. *We say that a valuation* ν *over* $\mathcal{L}$ satisfies (is a solution to) *GCSP* $\mathcal{C}$ *when* ν *is a solution to every constraint in* $\mathcal{C}$.

For example, any subset of set $\{c_2, c_3, c_4\}$ of constraints forms a GCSP.

From GCSP to Constraint Satisfaction Problem 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 $\mathcal{L}_1$ and $\mathcal{L}_2$ are finite-domain. Consider a special case of a constraint of the form (4) over finite-domain lexicon $\mathcal{L} = ([\Sigma, D], \rho)$ 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 syntactically identify (4) with the pair

$$\langle (t_1, \dots, t_n), P^{\rho} \rangle. \tag{6}$$

A *constraint satisfaction problem* (CSP) is a set of pairs (6), where $\Sigma_{|v}$ and D of the finite-domain lexicon $\mathcal{L}$ are called variables and domain of CSP, respectively. Saying that valuation ν over $\mathcal{L}$ satisfies (4) is the same as saying that $\langle t_1^{\nu}, \dots, t_n^{\nu} \rangle \in P^{\rho}$. The latter is the way in which a solution to expressions (6) 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 (6) in C. In conclusion, GCSP generalizes CSP by (i) elevating the restriction of finite-domain, and (ii) allowing us more elaborate syntactic expressions (e.g., recall f-symbols).

Constraint Answer Set Programs and Constraint Formulas Let σ_r and σ_i be two disjoint vocabularies. We refer to their elements as *regular* and *irregular* atoms respectively. For a program Π, by $At(\Pi)$ we denote the set of atoms occurring in it. Similarly, for a propositional formula F, by $At(F)$ we denote the set of its atoms.

Definition 4 *A* constraint answer set program *(CAS program) over the vocabulary* $\sigma = \sigma_r \cup \sigma_i$ *is a triple* $\langle \Pi, \mathcal{B}, \gamma \rangle$, *where* Π *is a logic program over the vocabulary* σ *such that* $hd(\Pi) \cap \sigma_i = \emptyset$, $\mathcal{B}$ *is a set of constraints over the same lexicon, and* γ *is an injective function from the set* σ_i *of irregular atoms to the set* $\mathcal{B}$ *of constraints.*

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

- $X \subseteq At(\Pi)$
- X *is an input answer set of* Π *relative to* σ_i, *and*
- *the following GCSP over* $\mathcal{L}$ *has a solution*

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

Note that $\neg\gamma(a)$ *may result in expression of the form* $\neg\neg P(t_1, \dots, t_n)$ *that we identify with* $P(t_1, \dots, t_n)$. *(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.

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

Definition 5 *A* constraint formula *over the vocabulary* $\sigma = \sigma_r \cup \sigma_i$ *is a triple* $\langle F, \mathcal{B}, \gamma \rangle$, *where* F *is a propositional formula over the vocabulary* σ, $\mathcal{B}$ *is a set of constraints over the same lexicon, and* γ *is an injective function from the set* σ_i *of irregular atoms to the set* $\mathcal{B}$ *of constraints.*

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

- $X \subseteq At(F)$
- *X is a model of F, and*
- *the following GCSP over $\mathcal{L}$ has a solution*

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

Following theorem captures a relation between CAS programs and constraint formulas.

Theorem 2 *For a CAS program $P = \langle \Pi, \mathcal{B}, \gamma \rangle$ 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 $\langle IComp(\Pi, \sigma_i), \mathcal{B}, \gamma \rangle$ over $\sigma = \sigma_r \cup \sigma_i$.*

Satisfiability Modulo Theories versus Constraint Formulas

First, in this section we introduce the notion of a "theory" in Satisfiability Modulo Theories (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. The presented notions of interpretation and satisfaction are usual, but are stated in terms convenient for our purposes. This facilitates uncovering the precise link between CASP-like formalisms and SMT-like formalisms. We note that in literature on SMT, the term "object constant" or "function symbol of arity 0" is commonly used to refer to elements in the signature that we call variables.

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

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

For signatures that contains no f-symbols, we omit the reference to the last element of the interpretation tuple.

For a signature Σ, a Σ-*theory* is a set of interpretations over Σ. For instance, for signature Σ_1, by $\mathcal{I}_1$ and $\mathcal{I}_2$ we denote the following sample interpretations (D_1, ν_1, ρ_1) and (D_1, ν_2, ρ_1) respectively. Any subset of interpretations $\{\mathcal{I}_1, \mathcal{I}_2\}$ exemplifies a unique Σ_1-theory.

A *restriction formula* over signature Σ is a finite set of constraint literals over c-vocabulary $[\Sigma, \emptyset]$. Consider a Σ-interpretation $I = (D, \nu, \rho, \phi)$. To each term τ over a c-vocabulary $[\Sigma, \emptyset]$, I assigns a value $\tau^{\nu,\phi}$ that we denote by τ^I. We say that I *satisfies* restriction formula Φ over Σ when ν satisfies every constraint literal in Φ w.r.t. ϕ and ρ. For instance, a sample restriction formula over Σ_1 follows

$$\{\neg E(s), \neg Q(r, s)\}. \quad (7)$$

Interpretation $\mathcal{I}_2$ satisfies this formula, while $\mathcal{I}_1$ does not.

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 (7) is satisfiable in any Σ_1-theory that contains interpretation $\mathcal{I}_2$. On the other hand, restriction formula (7) is not satisfiable in Σ_1-theory $\{\mathcal{I}_1\}$.

SMT and ASPT Programs We now introduce SMT formulas that merge the concepts of propositional formulas and Σ-theories. Then, we present ASPT programs that merge the concepts of logic programs and Σ-theories.

Definition 6 *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 $X \subseteq \sigma$ is its model *if*

- *$X \subseteq At(F)$,*
- *X is a model of F, and*
- *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 and Tinelli, 2014, Section 1.1], which, in their most commonly used case, coincide with SMT formulas discussed here.

Definition 7 *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, 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 $X \subseteq \sigma$ is its model *if*

- *$X \subseteq At(\Pi)$,*
- *X is an input answer set of Π relative to σ_i, and*
- *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.

Uniform Theories The presented definition of a Σ-theory 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 so called "uniform" theories that follow these typical restrictions. We will then show how restriction formulas interpreted over uniform theories can practically be seen as syntactic variants of GCSPs. This connection brings us to a straightforward relation between SMT formulas over uniform theories and constraint formulas as well as between CAS programs and ASPT programs. In the following section, we list several common SMT fragments such as satisfiability modulo difference logic and satisfiability modulo linear arithmetic whose theories are, in fact, uniform theories. We then use these findings to relate several ASP modulo theories approaches such as ASP(DL) introduced in [Liu *et al.*, 2012] and ASP(LC) introduced in [Liu *et al.*, 2012] to CASP approaches.

Definition 8 *For a signature Σ, we call a Σ-theory T* uniform *over lexicon $\mathcal{L} = ([\Sigma, D], \rho, \phi)$ when (i) all interpretations in T are of the form (D, ν, ρ, ϕ) (note how valuation ν is the only not fixed element in the interpretations), and (ii) for every possible $[\Sigma_{|v}, D]$ valuation ν, there is an interpretation (D, ν, ρ, ϕ) in T.*

To illustrate a concept of a uniform theory, a table below defines sample domain D_2, valuations ν_3 and ν_4, and r-denotation ρ_3.

D_2	$\{1, 2\}$
ν_3	$[\Sigma_{1\|v}, D_2]$ valuation, where $s^{\nu_3} = 1$ and $r^{\nu_3} = 2$
ν_4	$[\Sigma_{1\|v}, D_2]$ valuation, where $s^{\nu_4} = r^{\nu_4} = 2$
ρ_3	$[\Sigma_{1\|r}, D_2]$ r-denotation, where $E^{\rho_3} = \{\langle 2\rangle\}, Q^{\rho_3} = \{\langle 1, 1\rangle, \langle 2, 2\rangle\}$

Valuations ν_1 and ν_2 can be seen not only as $[\Sigma_{1|v}, D_1]$ valuations but also as $[\Sigma_{1|v}, D_2]$ valuations. The set

$$\{(D_2, \nu_1, \rho_3), (D_2, \nu_2, \rho_3), (D_2, \nu_3, \rho_3), (D_2, \nu_4, \rho_3)\}$$

of Σ_1 interpretations is an example of a uniform theory over lexicon $([\Sigma_1, D_2], \rho_3)$. We denote this theory by T_1. On the other hand, the set

$$\{(D_2, \nu_1, \rho_3), (D_2, \nu_2, \rho_3), (D_2, \nu_3, \rho_3), (D_1, \nu_4, \rho_3)\}$$

of Σ_1 interpretations is an example of a non-uniform theory. Indeed, the condition (i) of Definition 8 does not hold for this theory: the last interpretation refers to a different domain than the others. Also, neither of Σ_1-theories $\{\mathcal{I}_1\}$, $\{\mathcal{I}_1, \mathcal{I}_2\}$ is uniform over lexicon $([\Sigma_1, D_1], \rho_1)$. In this case, the condition (ii) of Definition 8 does not hold.

It is easy to see that for uniform theories we can identify their interpretations of the form (D, ν, ρ, ϕ) with their second element valuation ν. 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, ν_1, ρ_3) of uniform theory T_1 as ν_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. In the following for the uniform theories we assume such a more general syntax. Similarly, 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 μ-mapping refers do the domain of lexicon now in place of an empty set in the earlier definition. The definition of ASPT program can be extended in the same style. For the case of uniform theories we will assume the definition of SMT formulas as stated in this paragraph. The same applies to the case of ASPT programs modulo uniform theories.

We now present a theorem that makes the connection between GCSPs over some lexicon $\mathcal{L}$ and restriction formulas interpreted using the uniform theory T over the same lexicon $\mathcal{L}$ apparent: the question whether a given GCSP over $\mathcal{L}$ has a solution translates into the question whether the set of constraint literals of 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. We then relate SMT formulas "modulo uniform theories" and constraint formulas, as well as ASPT programs and CAS programs.

Theorem 3 *For a lexicon $\mathcal{L} = ([\Sigma, D], \rho, \phi)$, a set Φ of constraint literals over c-vocabulary $[\Sigma, D]$, a uniform Σ-theory T over lexicon $\mathcal{L}$, the following holds*

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

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

Theorem 4 *For a lexicon $\mathcal{L} = ([\Sigma, D], \rho, \phi)$, a vocabulary $\sigma = \sigma_r \cup \sigma_i$, and a set X of atoms over σ, set X is a model of SMT formula $\langle F, T_{\mathcal{L}}, \mu\rangle$ over σ if and only if X is a model of a constraint formula $\langle F, \mathcal{B}_{\mathcal{L}}, \mu\rangle$ over σ (where μ is identified with the function from irregular atoms to constraints over $\mathcal{L}$ in a trivial way.)*

This theorem illustrates that for uniform theories the language of SMT formulas and constraint formulas coincide. Or, that the language of constraint formulas is a special case of SMT formulas that are defined over uniform theories. We now show similar relation between CAS and ASPT programs.

Theorem 5 *For a lexicon $\mathcal{L} = ([\Sigma, D], \rho, \phi)$, a vocabulary $\sigma = \sigma_r \cup \sigma_i$, and a set X of atoms over σ, set X is a model of ASPT program $\langle \Pi, T_{\mathcal{L}}, \mu\rangle$ over σ if and only if X is a model of a CAS formula $\langle \Pi, \mathcal{B}_{\mathcal{L}}, \mu\rangle$ over σ.*

SMT and CASP Connection

This section starts by introducing numeric signatures and lexicons, and particular uniform theories. These definitions allow us to precisely define the languages used by various constraint answer set solvers. We conclude with the discussion of the variety of solving techniques used in logic programming community.

Let $\mathbb{Z}$ and $\mathbb{R}$ denote the sets of integers and real numbers respectively. We say that a signature is *numeric* when it satisfies the following requirements (i) its only f-symbols are $+$, $\times$ of arity 2, and (ii) its only predicate symbols are $<, >, \leq, \geq, =, \neq$ of arity 2. We use the symbol $\mathcal{A}$ to denote a numeric signature. Let $\phi_{\mathbb{Z}}$ and $\rho_{\mathbb{Z}}$ be $[\{+, \times\}, \mathbb{Z}]$ f-denotation and $[\{<, >, \leq, \geq, =, \neq\}, \mathbb{Z}]$ r-denotation respectively, where they map their function and predicate symbols into usual arithmetic operations and relations over integers. We call any lexicon of the form $([\mathcal{A}, \mathbb{Z}], \rho_{\mathbb{Z}}, \phi_{\mathbb{Z}})$ *integer*. Similarly, $\phi_{\mathbb{R}}$ and $\rho_{\mathbb{R}}$ denote $[\{+, \times\}, \mathbb{R}]$ f-denotation and

$[\{<,>,\leq,\geq,=,\neq\},\mathbb{R}]$ r-denotation respectively, where they map their function and predicate symbols into usual arithmetic operations and relations over reals. We call any lexicon of the form $([\mathcal{A},\mathbb{R}],\rho_{\mathbb{R}},\phi_{\mathbb{R}})$ *numeric*.

Such commonly used theories in SMT as *linear real arithmetic*, *linear integer arithmetic*, and *integer difference logic* are uniform. To be more precise, 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 integer linear arithmetic posing yet additional syntactic restrictions [Nieuwenhuis and Oliveras, 2005].

We call any ASPT program $\langle\Pi,T,\mu\rangle$ over $\sigma_r\cup\sigma_i$

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

In the same style, we can define SMT(L), SMT(IL), and SMT(DL) formulas.

From Theorem 5, CAS programs of the form $\langle\Pi,\mathcal{B}^*_{\mathcal{L}},\gamma\rangle$, where $\mathcal{L}$ is a numeric lexicon and $\mathcal{B}^*_{\mathcal{L}}$ is the set of all linear constraints over $\mathcal{L}$, are essentially the same objects as ASPT(L) programs. Similarly, it follows that CAS programs of the form $\langle\Pi,\mathcal{B}^*_{\mathcal{L}},\gamma\rangle$, where $\mathcal{L}$ is an integer lexicon and $\mathcal{B}^*_{\mathcal{L}}$ is the set of all integer linear constraints over $\mathcal{L}$, are essentially the same objects as ASPT(IL) programs.

Obviously, Theorems 2 and 4 pave the way for using SMT systems that solve SMT(L) and SMT(IL) problems as is for solving tight ASPT(L) and ASPT(IL) programs respectively. It is sufficient to compute the input completion of the program relative to irregular atoms. This observation has been utilized 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. It is a direction of future work to generalize these results to arbitrary theories.

Outlook on Constraint Answer Set Solvers Table 1 presents the landscape of current constraint answer set solvers using the unified terminology of this section. 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).

Solver	Language
CLINGCON [Gebser *et al.*, 2009]	ASPT(IL)*
EZCSP [Balduccini, 2009]	ASPT(IL)* ASPT(IL) ASPT(L)
MINGO [Liu *et al.*, 2012]	ASPT(L)
DINGO [Janhunen *et al.*, 2011]	ASPT(DL)

Table 1: Solvers Categorization

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 $\langle\Pi,\mathcal{B},\gamma\rangle$, both CLINGCON and EZCSP first use an answer set solver to compute an input answer set of Π. Second, they contact a theory solver to verify whether respective constraint satisfaction problem 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(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 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 CLINGON 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 typically 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) programs, hints that a similar strategy can be implemented for solving SMT(L) formulas. These ideas have recently been explored in [King *et al.*, 2014].

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 so called uniform theories widely used in SMT practice. Given such theories, CASP and SMT solving share more in common than meets the eye. 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. In the future, we would like to investigate a similar link to a related formalism of HEX-programs [Eiter *et al.*, 2012].

References

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

[Barrett and Tinelli, 2014] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund Clarke, Tom Henzinger, and Helmut Veith, editors, *Handbook of Model Checking*. Springer, 2014.

[Carlsson and Fruehwirth, 2014] Mats Carlsson and Thom Fruehwirth. *Sicstus PROLOG User's Manual 4.3*. Books On Demand - Proquest, 2014.

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[De Moura and Bjørner, 2008] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[Eiter *et al.*, 2012] Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. Conflict-driven ASP solving with external sources. *TPLP*, 12(4-5):659–679, 2012.

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

[Gebser *et al.*, 2009] Martin Gebser, Max Ostrowski, and Torsten Schaub. Constraint answer set solving. In *Proceedings of 25th International Conference on Logic Programming (ICLP)*, pages 235–249. Springer, 2009.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

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

[Janhunen *et al.*, 2011] Tomi Janhunen, Guohua Liu, and Ilkka Niemela. Tight integration of non-ground answer set programming and satisfiability modulo theories. In *Proceedings of the 1st Workshop on Grounding and Transformations for Theories with Variables*, 2011.

[King *et al.*, 2014] Tim King, Clark Barrett, and Cesare Tinelli. Leveraging linear and mixed integer programming for smt. In *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, FMCAD '14, pages 24:139–24:146, Austin, TX, 2014. FMCAD Inc.

[Lee and Meng, 2013] Joohyung Lee and Yunsong Meng. Answer set programming modulo theories and reasoning about continuous changes. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13), Beijing, China, August 3-9, 2013*, 2013.

[Lierler and Truszczynski, 2011] Yuliya Lierler and Miroslaw Truszczynski. Transition systems for model generators — a unifying approach. *Theory and Practice of Logic Programming, 27th International Conference on Logic Programming (ICLP'11) Special Issue*, 11, issue 4-5, 2011.

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

[Liu *et al.*, 2012] Guohua Liu, Tomi Janhunen, and Ilkka Niemela. Answer set programming via mixed integer programming. In *Knowledge Representation and Reasoning Conference*, 2012.

[Marriott and Stuckey, 1998] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.

[Mellarkod *et al.*, 2008] Veena S. Mellarkod, Michael Gelfond, and Yuanlin Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1):251–287, 2008.

[Nieuwenhuis and Oliveras, 2005] Robert Nieuwenhuis and Albert Oliveras. Dpll(t) with exhaustive theory propagation and its application to difference logic. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05), volume 3576 of LNCS*. Springer, 2005.

[Wielemaker *et al.*, 2012] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.

[Zhou, 2012] Neng-fa Zhou. The language features and architecture of b-prolog. *Theory and Practice of Logic Programming*, 12(1-2):189–218, January 2012.