ICLP Tutorial: Relating Constraint Answer Set Programming and Satisfiability Modulo Theories

Yuliya Lierler

Available at: https://works.bepress.com/yuliya_lierler/68/
Relating Constraint Answer Set Programming and Satisfiability Modulo Theories

Yuliya Lierler*

University of Nebraska at Omaha

*Thanks to Ben Susman
Outline

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
## Two Main Subjects of Discourse for this Tutorial

### Constraint Answer Set Programming (CASP)
- Roots from **Answer Set Programming (ASP)**
- Declarative programming with an emphasis on modeling with high performance
- Incorporates **constraint processing**, avoiding grounding blow-up
- Solvers such as ACSOLVER, CLINGCON, EZCSP, INCA, IDP

### Satisfiability Modulo Theories (SMT)
- Roots from **propositional satisfiability (SAT)**
- Incorporates specialized **theory solving**
- Solvers such as cvc4 and z3

### Relation between CASP and SMT
- Informally: obviously related
- Formally: *who knows* exact relation and do we have to know?
It Gets More Complex

Constraint Answer Set Programming (CASP)
- Roots from ASP
- Incorporates constraint processing, avoiding grounding blow-up
- Solvers such as ACSOLVER, CLINGCON, EZCSP, INCA, IDP

Answer Set Programming Modulo Theories
- Roots from ASP
- Incorporates specialized theory solving as in SMT
  - Logic programs modulo linear constraints and solver MINGO, Liu et.al. 2012
  - Logic programs modulo difference constraints and solver DINGO, Janhunen et.al. 2011
  - ASPMT programs and solver system ASPMT2SMT, Bartholomew, Lee, and Meng 2013, 2014

Relation between CASP and ASP Modulo Theories
Formally: who knows exact relation. Yet, it seems that in this case we have to know.
Relation between CASP and ASP Modulo Theories

Knowing Formal Link Allows Us To Answer Questions

- Does logic program written in a CASP language mean the same as a logic program written in ASP modulo theories?
- Or, in other words, are these the same or different paradigms?
- What is exact link between SMT and CASP?
- Multitude of solvers: can ideas behind one or another solver be used in a "different" paradigm? and what to expect?
- What can SMT borrow from CASP? What can CASP borrow from SMT?

These Questions as Motivation

- This tutorial provides details on formal link between SMT, CASP, and ASP modulo theories paradigms.
- We also overview the existing technological solutions to solving in these paradigms and provide experimental details.
Plan

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
CASP vs SMT informally: Syntactically

**CASP program**

Composed of rules

\[ a_0 \leftarrow a_1, \ldots a_m, \text{not } a_{m+1}, \ldots \text{not } a_n, \]

where \(a_i\) is an atom over a vocabulary \(\sigma_r \cup \sigma_i\):

- \(\sigma_r\) are regular atoms and
- \(\sigma_i\) are irregular atoms.

**SMT formula**

Composed of clauses

\[ l_0 \lor \cdots \lor l_n, \]

where \(l_i\) is a literal over a vocabulary \(\sigma_r \cup \sigma_i\):

- \(\sigma_r\) are regular atoms and
- \(\sigma_t\) are theory atoms.
CASP vs SMT informally: Semantically

**CASP program**

A set $X$ of atoms over $\sigma_r \cup \sigma_i$ is a *solution* to CASP program $\Pi$ when

1. $X$ is an answer set of $\Pi$ seen as a regular program (modulo slightly special treatment of irregular atoms)
2. constraint satisfaction problem constructed from $X$ using the "true" meaning of $\sigma_i$ participants of $X$.

**SMT formula**

A set $M$ of literals over $\sigma_r \cup \sigma_t$ is a *solution* to SMT formula $F$ when

1. $M$ is a model of $F$ seen as a regular propositional logic formula
2. first order theory constructed from $M$ using the "true" meaning of $\sigma_t$ participants of $M$ is $T$-satisfiable.
CASP vs SMT informally: Semantically

CASP program vs SMT formula: the first condition

A set $X$ of atoms over $\sigma_r \cup \sigma_i$ is a solution to CASP program $\Pi$ when

1. $X$ is an answer set of $\Pi$ seen as a regular program (modulo slightly special treatment of irregular atoms)

A set $M$ of literals over $\sigma_r \cup \sigma_t$ is a solution to SMT formula $F$ when

1. $M$ is a model of $F$ seen as a regular propositional logic formula

A Relation between SAT and Logic Programs is Well Understood

- From SAT to logic programs: trivial
- From logic program to SAT:
  - Completion (tight programs)
  - Completion plus loop formulas (arbitrary programs)
  - Other translations that extend the original vocabulary of a program

A number of answer set solvers rely on these translations and form so called SAT-based group of solvers (e.g., ASSAT, CMODELS, LPR2SAT).
CASP vs SMT informally: Semantically

CASP program vs SMT formula: the second condition

A set $X$ of atoms over $\sigma_r \cup \sigma_i$ is a solution to CASP program $\Pi$ when

2. constraint satisfaction problem constructed from $X$ using the "true" meaning of $\sigma_i$ participants of $X$.

A set $M$ of literals over $\sigma_r \cup \sigma_t$ is a solution to SMT formula $F$ when

2. $X$ is an answer set of $\Pi$ seen as a regular program (modulo slightly special treatment of irregular atoms)

?: Constraint Satisfaction Problems vs T-satisfiable Theories

- This is the question the tutorial will focus on
- The main complexity of the question is terminological
- Constraint Satisfaction Problems are studied in Constraint Programming automated reasoning subfield
- T-satisfiable theories are studied in SMT automated reasoning subfield
CASP as a Programming Framework

- CASP languages (similarly to ASP languages) allow for non-constraint variables.
- These variables are then eliminated in the process of grounding to form "propositional" programs.
- Thus, CASP language gives us fine modeling capabilities for encoding problems.
- CASP is a programming paradigm.
- There is no counterpart to that within SMT.
  - The SMT-LIB language of SMT:
    - forms a standard interface for solvers.
    - but is not meant for programming.
    - SMT-LIB specifications are assumed to be machine producible.
Grounding provides basis for Programming Framework

Consider a sample CASP rule in the language of CLINGCON-2:

\[ \text{:- } \text{volume}(B,T+1) \neq \text{volume}(B,T) + A, \text{pour}(B,T,A). \]

If suitable terms for \( B \), \( T \), and \( A \) range over \([a], [0, 1], \) and \([1, 2] \) respectively, then this rule is instantiated while grounding as follows:

\[ \text{:- } \text{volume}(a,1) \neq \text{volume}(a,0) + 1, \text{pour}(a,0,1). \]
\[ \text{:- } \text{volume}(a,2) \neq \text{volume}(a,1) + 1, \text{pour}(a,1,1). \]
\[ \text{:- } \text{volume}(a,1) \neq \text{volume}(a,0) + 2, \text{pour}(a,0,2). \]
\[ \text{:- } \text{volume}(a,2) \neq \text{volume}(a,1) + 2, \text{pour}(a,1,2). \]

SMT-LIB limitation

Note: The SMT-LIB language only allows us to code in the later form (modulo differences in syntax.)

Yet, if suitable terms for \( B \), \( T \), and \( A \) range over \( k, m, \) and \( n \) variables respectively, then a grounder produces \( k \times m \times n \) rules.

Say, \( k = 5, m = 10, and n = 10 \): we land at 500.
Plan

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
Generalized Constraint Satisfaction Problem in a Nutshell

- Syntactically, a constraint is practically a ground literal from predicate logic.
- Generalized constraint satisfaction problem or GCSP is a tuple containing:
  - set of constraints
  - domain or universe, say $D$
  - "denotations" for function symbols (of non-0 arity) and predicate symbols:
    - denotation for an $n$-ary function symbol is a mapping to a function $D^n \rightarrow D$
    - denotation for an $n$-ary predicate symbol is a mapping to an $n$-ary relation on $D$ (i.e., a subset of the Cartesian product $D^n$)
- A solution to GCSP is a valuation for object constants (a mapping from object constants to domain values) that satisfy every constraint in GCSP.
A domain $D$ is nonempty set of elements

A signature $\Sigma$ consisting of
- variables $V$
- predicate symbols $R$
- function symbols $F$

A function $\nu : V \to D$ is referred to as a $[\Sigma, D]$ valuation

A function $\rho$ mapping $n$-ary predicate symbols in $R$ into $n$-ary relations is a $[\Sigma, D]$ $r$-denotation

A function $\phi$ mapping $n$-ary function symbols in $F$ to $D^n \mapsto D$ functions is a $[\Sigma, D]$ $f$-denotation

Example

$\Sigma = \{s, r, E/1, Q/2\}$ (s, r are variables; E, Q are predicate symbols)

$D = \{1, 2\}$

$\nu [\Sigma, D]$ valuation, where $s^\nu = r^\nu = 1$

$\rho [\Sigma, D]$ r-denotation, where $E^\rho = \{\langle 1 \rangle\}$, $Q^\rho = \{\langle 1, 1 \rangle, \langle 2, 2 \rangle\}$
A *constraint vocabulary* is a pair $[\Sigma, D]$

A *term* over a constraint vocabulary is either

- A variable in $\Sigma$
- A domain element in $D$
- An expression $f(t_1, \ldots, t_n)$, where $f$ is $n$-ary function symbol in $\Sigma$ and $t_1, \ldots, t_n$ are terms

A *constraint literal* is a predicate symbol $P$ from $\Sigma$ of $n$-arity of the form

$$P(t_1, \ldots, t_n) \quad \text{or} \quad \neg P(t_1, \ldots, t_n)$$

where $t_1, \ldots, t_n$ are terms.

**Example**

<table>
<thead>
<tr>
<th>$\Sigma$</th>
<th>${s, r, E, Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>${1, 2}$</td>
</tr>
<tr>
<td>Constraint Literals over $[\Sigma, D]$</td>
<td>$\neg E(s), \neg E(2), \text{and } Q(r, s)$</td>
</tr>
</tbody>
</table>
A lexicon is a tuple ([Σ, D], ρ, φ), where ρ and φ are [Σ, D] r-and f-denotations.

Example

<table>
<thead>
<tr>
<th>Σ</th>
<th>{s, r, E, Q}</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>ρ</td>
<td>[Σ, D] r-denotation, where ( E^ρ = {\langle 1 \rangle} ), ( Q^ρ = {\langle 1, 1 \rangle, \langle 2, 2 \rangle} )</td>
</tr>
<tr>
<td>Lexicon</td>
<td>([Σ, D], ρ)</td>
</tr>
<tr>
<td></td>
<td>(we omit listing φ when no function symbols are present in Σ)</td>
</tr>
</tbody>
</table>

Constraints: Syntactically

A constraint is defined over a lexicon ([Σ, D], ρ, φ):

**Syntactically:**

it is a constraint literal over [Σ, D] (or, over a lexicon ([Σ, D], ρ, φ))

Example

Constraint literals over [Σ, D] \( ¬E(s), ¬E(2), \) and \( Q(r, s) \)
Valuation and f-denotation for Evaluating Terms Semantically

Given

- term $\tau$ over constraint vocabulary $[\Sigma, D]$
- $[\Sigma, D]$ valuation $\nu$
- $[\Sigma, D]$ f-denotation $\phi$

We say that $\nu$ assigns a value to $\tau$ w.r.t. $\phi$ as follows:

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

$$f(t_1, \ldots, t_n)^{\nu, \phi} = f^{\phi}(t_1^{\nu, \phi}, \ldots, t_n^{\nu, \phi}).$$
Constraints, Semantically

**Syntactically:** a constraint is a constraint literal

\[ P(t_1, \ldots, t_n) \text{ or } \neg P(t_1, \ldots, t_n) \]

over a lexicon \([\Sigma, D], \rho, \phi\)

**Semantically:** \([\Sigma, D]\) valuation \(\nu\) is a *solution* for constraint

\[
\begin{align*}
P(t_1, \ldots, t_n) & \quad \text{if } \langle t_1^{\nu,\phi}, \ldots, t_n^{\nu,\phi} \rangle \in P^\rho \\
\neg P(t_1, \ldots, t_n) & \quad \text{if } \langle t_1^{\nu,\phi}, \ldots, t_n^{\nu,\phi} \rangle \notin P^\rho
\end{align*}
\]

**Example**

<table>
<thead>
<tr>
<th>(\Sigma) and (D)</th>
<th>{s, r, E, Q} and {1, 2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\rho)</td>
<td>([\Sigma, D]) r-denotation, where (E^\rho = {\langle 1 \rangle}, \ Q^\rho = {\langle 1, 1 \rangle, \langle 2, 2 \rangle})</td>
</tr>
<tr>
<td>(L)</td>
<td>([\Sigma, D], \rho)</td>
</tr>
</tbody>
</table>

Sample constraints

- literal \(Q(r, s)\) over \(L\); literal \(\neg E(1)\) over \(L\)

<table>
<thead>
<tr>
<th>(\nu)</th>
<th>([\Sigma, D]) valuation, where (r^\nu = s^\nu = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q(r, s))</td>
<td>(\langle r^\nu, s^\nu \rangle = \langle 1, 1 \rangle) in (Q^\rho = {\langle 1, 1 \rangle, \langle 2, 2 \rangle}): (\nu) is a solution</td>
</tr>
<tr>
<td>(\neg E(1))</td>
<td>(\langle 1^\nu \rangle = \langle 1 \rangle) in (E^\rho = {\langle 1 \rangle}): (\nu) is not a solution</td>
</tr>
</tbody>
</table>
**Defining GCSP**

*Generalized constraint satisfaction problem* (GCSP) is a set of constraints over the same lexicon.

Valuation $\nu$ is a *solution* to a GCSP when $\nu$ is a solution to all of its constraints.

### Example

<table>
<thead>
<tr>
<th>$\Sigma$ and $D$</th>
<th>${s, r, E, Q}$ and ${1, 2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho$</td>
<td>$[\Sigma, D]$ r-denotation, where $E^\rho = {\langle 1 \rangle}$, $Q^\rho = {\langle 1, 1 \rangle, \langle 2, 2 \rangle}$</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>$([\Sigma, D], \rho)$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>literal $Q(r, s)$ over $\mathcal{L}$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>literal $\neg E(1)$ over $\mathcal{L}$</td>
</tr>
<tr>
<td>Sample GCSPs</td>
<td>${c_1}$ and ${c_1, c_2}$</td>
</tr>
<tr>
<td>$\nu$</td>
<td>$[\Sigma, D]$ valuation, where $s^\nu = r^\nu = 1$</td>
</tr>
</tbody>
</table>

Note $\nu$ is a solution to the GCSP $\{c_1\}$, but not $\{c_1, c_2\}$.
Numeric Lexicons

In practice, GCSP of particular kind are often used: these that encode linear inequalities and integer linear inequalities

**Numeric Signature**

A *numeric signature* is a signature that satisfies the following requirements

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

**Numeric and Integer Lexicons**

$\rho_R$ and $\rho_Z$ map $\{<, >, \leq, \geq, =, \neq\}$ into arithmetic relations over reals and over integers.

$\phi_R$ and $\phi_Z$ map f-symbols $\{+ , \times\}$ into arithmetic functions over reals and over integers.

For a numeric signature $\mathcal{A}$

- a lexicon of the form $(\mathcal{A}, \mathbb{R}, \rho_R, \phi_R)$ is called *numeric*
- a lexicon of the form $(\mathcal{A}, \mathbb{Z}, \rho_Z, \phi_Z)$ is called *integer*
Linear expressions

A *numeric linear expression*:

\[ a_1x_1 + \cdots + a_nx_n, \]  

(1)

\(a_1, \ldots, a_n\) are real numbers and \(x_1, \ldots, x_n\) are variables over real numbers.

This expression is an abbreviation for the following term

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

over some numeric lexicon \((\mathcal{A}, \mathbb{R}, \rho_\mathbb{R}, \phi_\mathbb{R})\), where \(\mathcal{A}\) contains \(x_1, \ldots, x_n\) as its variables.

**Example**

\(2x + 3y\) stands for \(+ (\times(2, x), \times(3, y))\)

Similarly, we define *integer linear expression* \((\text{real} \rightarrow \text{integer})\).
Linear GCSP

Given

- $e$ a linear expression
- $k$ a real number
- $\triangleright$ an element from $\{<, >, \leq, \geq, =, \neq\}$.

A constraint is linear when it is defined over some numeric lexicon (written in usual infix notation)

$$e \triangleright k$$

$2x + 3y > 5$ is a sample linear constraint. (Implicitly, assume numeric lexicon to contain variable $x, y$ in its signature.)

A GCSP consisting of linear constraints is a linear constraint satisfaction problem (L-CSP).

L-CSP

- $\{x > 4, x < 5\}$
- $\{2x + 3y > 5, x > 4, x < 5\}$

infinite number of solutions include $x = 4.1$

infinite number of solutions include $x = 4.1, y = 0$
Integer Linear GCSP

Given

- $e$ an integer linear expression
- $k$ an integer
- $\bowtie$ an element from $\{<, >, \leq, \geq, =, \neq\}$.

A constraint is *integer linear* when it is defined over some *integer* lexicon

$$e \bowtie k$$

$2x + 3y > 5$ is a sample integer constraint.

A GCSP consisting of integer linear constraints is a *integer linear constraint satisfaction problem (IL-CSP)*.

**IL-CSP**

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2x + 3y &gt; 5$</td>
<td>infinite number of solutions include $x = 2, y = 1$</td>
</tr>
<tr>
<td>${x &gt; 4}$</td>
<td>infinite number of solutions include $x = 5$</td>
</tr>
<tr>
<td>${x &lt; 5}$</td>
<td>infinite number of solutions include $x = 4$</td>
</tr>
<tr>
<td>${x &gt; 4, x &lt; 5, 2x + 3y &gt; 5}$</td>
<td>no solutions.</td>
</tr>
</tbody>
</table>
Plan

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
Vocabulary is a set of atoms.

A logic program $\Pi$ over vocabulary $\sigma$ is a set of rules of the form

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

Informally, answer sets: sets of atoms that are seen as solutions to a program.

For a program $\Pi$ over $\sigma$, a set $X$ of atoms over $\sigma$ is an input answer set of $\Pi$ relative to vocabulary $\iota \subseteq \sigma$ when $X$ is an answer set of the program

$$\Pi \cup ((X \cap \iota) \setminus \text{heads}(\Pi)).$$

Example

<table>
<thead>
<tr>
<th>logic program</th>
<th>answer set</th>
<th>input answer set relative to input vocabulary ${\text{switch, am}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>lightOn $\leftarrow$ switch, not am. \leftarrow not lightOn.</td>
<td>$\emptyset$</td>
<td>${\text{switch, lightOn}}$</td>
</tr>
</tbody>
</table>
Defining Constraint Answer Set Programs

Given

\[
\sigma_r \cup \sigma_i \quad \text{vocabulary (regular atoms } \sigma_r \text{ and irregular atoms } \sigma_i \text{ are disjoint)} \\
\Pi \quad \text{a logic program over } \sigma_r \cup \sigma_i, \text{ where } \text{heads}(\Pi) \cap \sigma_i = \emptyset \\
\mathcal{B} \quad \text{set of constraints over lexicon } \mathcal{L} \\
\gamma \quad \text{an injective function } \sigma_i \to \mathcal{B}
\]

Constraint Answer Set program

A \textit{CAS program} \( P \) is a triple \( \langle \Pi, \mathcal{B}, \gamma \rangle \).

A set \( X \subseteq \sigma_r \cup \sigma_i \) is an \textit{answer set} of \( P \) if

- \( X \) is an input answer set of \( \Pi \) relative to \( \sigma_i \), and
- the following GCSP has a solution

\[
\{ \gamma(a) \mid a \in X \cap \sigma_i \} \cup \{ \neg \gamma(a) \mid a \in (\text{At}(\Pi) \cap \sigma_i) \setminus X \}.
\]
CAS Program Example

CAS Program $P_1 = \langle \Pi_1, B_1, \gamma_1 \rangle$ over $\sigma_r \cup \sigma_i$

- $\sigma_r = \{\text{lightOn, switch, am}\}$ and $\sigma_i = \{|x < 12|, |x \geq 12|, |x < 0|, |x > 23|\}$
- $B_1 = \{x < 12, x \geq 12, x < 0, x \geq 0, x > 23, x \leq 23\}$ over integer lexicon
- $\gamma_1(a) = \begin{cases} x < 12 & \text{if } a = |x < 12| \cdots ; \\ x \geq 12 & \text{if } a = |x < 12| \cdots \end{cases}$

$\Pi_1 : \{\text{lightOn} \leftarrow \text{switch, not am.} \}$
$\quad \leftarrow \text{not lightOn.}$
$\quad \{\text{switch}.\}$
$\quad \{\text{am}.\}$
$\quad \leftarrow \text{not am, } |x < 12|.$
$\quad \leftarrow \text{am, } |x \geq 12|.$
$\quad \leftarrow |x < 0|.$
$\quad \leftarrow |x > 23|.$

$\Pi_1 \quad \text{input answer set relative to } \sigma_i :$
$\quad \{\text{switch, lightOn, } |x \geq 12|\}$

IL-CSP $\{x \geq 12, x \geq 0, x \leq 23\}$ solutions: $x = 12$ or $\cdots$ or $x = 23$

$P_1 \quad \text{answer set } \{\text{switch, lightOn, } |x \geq 12|\}$
Defining Constraint Formulas

Recall: A CAS program $P$ is a triple $\langle \Pi, B, \gamma \rangle$.

A set $X \subseteq \sigma_r \cup \sigma_i$ is an answer set of $P$ if

- $X$ is an input answer set of $\Pi$ relative to $\sigma_i$, and
- the following GCSP has a solution

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

Input Completion ($I\text{Comp}(\Pi, \iota)$)

Definition similar to Clark completion.

For a program $\Pi$ over $\sigma$, the input-completion of $\Pi$ relative to vocabulary $\iota \subseteq \sigma$, $I\text{Comp}(\Pi, \iota)$, is the set of propositional formulas that consists of

- the rules in $\Pi$ understood as implications

$$a_0 \leftarrow a_1, \ldots a_m, \text{not } a_{m+1}, \ldots \text{not } a_n \iff a_1 \land \cdots \land a_m \land \neg a_{m+1} \land \cdots \land \neg a_n \to a_0$$

- for all atoms $a$ occurring in $(\sigma \setminus \iota) \cup \text{heads}(\Pi)$, the implications

$$a \to \bigvee_{a \leftarrow B \in \Pi} B.$$

Properties

- For tight programs, input answer sets coincide with models of input completion.
- Tight programs do not allow "positive cycles". A simple non-tight program:

$$p \leftarrow p.$$  
- Many practical programs are tight.
CAS Programs and Constraint Formulas Connection

For a tight program $\Pi$, the following statements are equivalent:

- $X$ is an answer set of $P = \langle \Pi, B, \gamma \rangle$ over the vocabulary $\sigma_r \cup \sigma_i$
- $X$ is a model of the constraint formula $\mathcal{F} = \langle IComp(\Pi, \sigma_i), B, \gamma \rangle$
Example: CAS Programs vs Constraint Formulas

CAS Program $P_1 = \langle \Pi_1, B_1, \gamma_1 \rangle$ over $\sigma_r \cup \sigma_i$

$\sigma_i = \{ |x < 12|, |x \geq 12|, |x < 0|, |x > 23| \}$

<table>
<thead>
<tr>
<th>tight $\Pi_1$</th>
<th>Formula equivalent to $I\text{Comp}(\Pi, \sigma_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{lightOn} \leftarrow \text{switch, not am}$</td>
<td>$\text{lightOn} \leftrightarrow \text{switch} \land \neg \text{am}$</td>
</tr>
<tr>
<td>$\leftarrow \text{not lightOn.} }$</td>
<td>$\text{lightOn}$</td>
</tr>
<tr>
<td>${ \text{switch} }.$</td>
<td>$\neg \text{switch} \lor \text{switch}$</td>
</tr>
<tr>
<td>${ \text{am} }.$</td>
<td>$\neg \text{am} \lor \text{am}.$</td>
</tr>
<tr>
<td>$\leftarrow \text{not am, }</td>
<td>x &lt; 12</td>
</tr>
<tr>
<td>$\leftarrow \text{am, }</td>
<td>x \geq 12</td>
</tr>
<tr>
<td>$\leftarrow</td>
<td>x &lt; 0</td>
</tr>
<tr>
<td>$\leftarrow</td>
<td>x &gt; 23</td>
</tr>
</tbody>
</table>

$I\text{Comp}(\Pi_1, \sigma_i)$

model :

$\{ \text{switch, lightOn, } |x \geq 12| \}$

IL-CSP $\{ x \geq 12, x \geq 0, x \leq 23 \}$ solutions: $x = 12$ or $\cdots$ or $x = 23$

$\langle I\text{Comp}(\Pi_1, \sigma_i), B_1, \gamma_1 \rangle$ model $\{ \text{switch, lightOn, } |x \geq 12| \}$
Plan

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
**SMT Theories**

### Σ-Interpretation and Σ-Theory

Usual notion of interpretation stemming from predicate logic:

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

- $D$ is a domain,
- $ν$ is a $[Σ, D]$ valuation
- $ρ$ is a $[Σ, D]$ r-denotation
- $φ$ is a $[Σ, D]$ f-denotation

A **Σ-theory** $T$ is a set of Σ-interpretation over $Σ$

### Sample Σ-theory

$Σ \{ s, r, E/1, Q/2 \}$

$D, D'$

$\{1, 2\}$ and $\{1, 2, 3\}$

$ν$ $[Σ, D]$ valuation, where $s^ν = r^ν = 1$

$ρ$ $[Σ, D]$ r-denotation, where $E^ρ = \{⟨1⟩\}$, $Q^ρ = \{⟨1, 1⟩, ⟨2, 2⟩\}$

$I$ $(D, ν, ρ)$

$I'$ $(D', ν, ρ)$

Any subset of interpretations $\{I, I'\}$ exemplifies a unique Σ-theory.
Restriction Formulas

Restriction formula

- **Syntactically**, a restriction formula is a finite set of constraint literals over \([\Sigma, \emptyset]\)
  - a conjunction of ground literals
- **Semantically**, a \(\Sigma\) interpretation \(I\) is a solution to a restriction formula over \([\Sigma, \emptyset]\)
  - if each of its literals is satisfied by \(I\)
  - satisfaction relation as in predicate logic

Sample restriction formula over \(\Sigma\)

\[
\begin{align*}
\Sigma & \quad \{s, r, E/1, Q/2\} \\
D & \quad \{1, 2\} \\
\nu & \quad [\Sigma, D] \text{ valuation, where } s^\nu = r^\nu = 1 \\
\rho & \quad [\Sigma, D] \text{ r-denotation, where } E^\rho = \{\langle 1 \rangle\}, \quad Q^\rho = \{\langle 1, 1 \rangle, \langle 2, 2 \rangle\} \\
\mathcal{I} & \quad (D, \nu, \rho)
\end{align*}
\]

\(\mathcal{I}\) is a solution for a sample restriction formula \(\{E(s), Q(r, s)\}\).

\(\mathcal{I}\) is not a solution for \(\{\neg E(s), \neg Q(r, s)\}\).
A restriction formula $\Phi$ over $\Sigma$ is **satisfiable** in a $\Sigma$-theory $T$, or is **$T$-satisfiable**, when there is an interpretation in $T$ that satisfies $\Phi$.

$\Sigma \{ s, r, E/1, Q/2 \}$

$D \{ 1, 2 \}$

$\nu$ is a $[\Sigma, D]$ valuation, where $s^{\nu} = r^{\nu} = 1$

$\rho$ is a $[\Sigma, D]$ r-denumeration, where $E^{\rho} = \{ \langle 1 \rangle \}$, $Q^{\rho} = \{ \langle 1, 1 \rangle, \langle 2, 2 \rangle \}$

$\mathcal{I} = (D, \nu, \rho)$

$\{ E(s), Q(r, s) \}$ is satisfiable in any $\Sigma$-theory containing $\mathcal{I}$, e.g., $\{ \mathcal{I} \}$.

$\{ \neg E(s), \neg Q(r, s) \}$ is not satisfiable in a $\Sigma$ theory $\{ \mathcal{I} \}$. 
Natural Question to Ask

Restriction formula vs GCSP

- Syntactically, identical to GCSP (modulo domain elements)
- Semantically?
Uniform Theories

A \( \Sigma \)-theory \( T \) is a *uniform theory* over lexicon \( \mathcal{L} = ([\Sigma, D], \rho, \phi) \) if

1. All interpretations in \( T \) are of the form \((D, \nu, \rho, \phi)\)
2. For every possible \([\Sigma, D]\) valuation \( \nu \), there is an interpretation \((D, \nu, \rho, \phi)\) in \( T \)

Note that \( D, \rho, \) and \( \phi \) are fixed among all interpretations:

Valuation \( \nu \) suffices to capture interpretations of uniform theories

\( GCSP = \text{restriction formula} \) interpreted with uniform theory "bind" by the lexicon of GCSP
Common SMT Theories are Uniform Theories

Sample Uniform Theories
- Integer linear arithmetic
- Linear arithmetic
- Difference logic

Thus
- IL-CSP = restriction formula in integer linear arithmetic theory
- L-CSP = restriction formula in linear arithmetic theory
Plan

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
Given

$\sigma_r \cup \sigma_i$ vocabulary

$F$ a propositional formula over $\sigma_r \cup \sigma_i$

$T$ $\Sigma$-theory

$\mu$ an injective function from $\sigma_i$ to constraint atoms over $\Sigma$

SMT

An \textit{SMT formula} $\mathcal{F}$ is a triple $\langle F, T, \mu \rangle$.

A set $X \subseteq \sigma_r \cup \sigma_i$ is a \textit{model} of $\mathcal{F}$ if

- $X$ is a model of $F$
- the following restriction formula is satisfiable in $\Sigma$-theory $T$

$$\{ \mu(a) | a \in X \cap \sigma_i \} \cup \{ \neg \mu(a) | a \in (\text{At}(F) \cap \sigma_i) \setminus X \}.$$
### Logic Programs Modulo Theories (ASPT)

**Given**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_r \cup \sigma_i$</td>
<td>vocabulary</td>
</tr>
<tr>
<td>$\Pi$</td>
<td>a logic program over $\sigma_r \cup \sigma_i$</td>
</tr>
<tr>
<td>$T$</td>
<td>$\Sigma$-theory</td>
</tr>
<tr>
<td>$\mu$</td>
<td>an injective function from $\sigma_i$ to constraint atoms over $\Sigma$</td>
</tr>
</tbody>
</table>

**ASPT**

An ASPT program $\mathcal{P}$ is a triple $\langle \Pi, T, \mu \rangle$.

A set $X \subseteq \sigma_r \cup \sigma_i$ is its answer set if

1. $X$ is an input answer set of $\Pi$ relative to $\sigma_i$
2. the following restriction formula is satisfiable in $\Sigma$-theory $T$

$$\{ \mu(a) | a \in X \cap \sigma_i \} \cup \{ \neg \mu(a) | a \in (At(F\Pi) \cap \sigma_i) \setminus X \}.$$
When uniform theories are under consideration:

\[ \text{SMT formulas} = \text{constraint formulas} \]
\[ \text{CAS programs} = \text{ASPT programs} \]

Some more notation

- an \textit{ASPT(IL) program} if \( T \) is integer linear arithmetic and \( \mu \) maps irregular atoms \( \sigma_i \) into integer linear constraints;
- an \textit{SMT(DL) formula} if \( T \) is difference logic and \( \mu \) maps irregular atoms \( \sigma_i \) into difference logic constraints;
- an \textit{SMT(L) formula} if \( T \) is linear arithmetic and \( \mu \) maps irregular atoms \( \sigma_i \) into linear constraints.
Plan

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
## CASP Systems

- ACSOLVER (Mellarkod, Gelfond, Zhang 2008)
- CLINGCON (Gebser, Ostrowski, Schaub 2009)
- EZCSP (Balduccini, Lierler 2009)
- INCA (Drescher, Walsh 2011)
- DINGO (Janhunen, Liu, Niemela 2011)
- MINGO (Liu, Niemela, Janhunen 2012)
- MINISAT (De Cat, Bogaerts, Denecker 2014)
- ASPMT2SMT (Bartholomew, Lee 2014)
- EZSMT (Susman, Lierler 2016)
### Solvers’ Categorization by Language

<table>
<thead>
<tr>
<th>Solver</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACSOLVER</td>
<td>ASPT(IL)*</td>
</tr>
<tr>
<td>CLINGCON</td>
<td>ASPT(IL)*</td>
</tr>
<tr>
<td>EZCSP</td>
<td>ASPT(IL)*</td>
</tr>
<tr>
<td></td>
<td>ASPT(IL)</td>
</tr>
<tr>
<td></td>
<td>ASPT(L)</td>
</tr>
<tr>
<td>INCA</td>
<td>ASPT(IL)*</td>
</tr>
<tr>
<td>DINGO</td>
<td>ASPT(DL)</td>
</tr>
<tr>
<td>MINGO</td>
<td>ASPT(L)+</td>
</tr>
<tr>
<td>MINISAT(ID)</td>
<td>ASPT(IL)+</td>
</tr>
<tr>
<td>ASPMT2SMT</td>
<td>ASPT(IL)**</td>
</tr>
<tr>
<td>EZSMT</td>
<td>ASPT(IL)</td>
</tr>
<tr>
<td></td>
<td>ASPT(L)</td>
</tr>
</tbody>
</table>

### Remarks

*: the solvers for this language require variables to be of finite ranges

+: the solver for this language allows variables to be either real or integer
### Solvers’ Categorization by Solving Technology

<table>
<thead>
<tr>
<th>Solving basis</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer set solver + CSP/CLP solver</td>
<td>ACSOLVER</td>
</tr>
<tr>
<td></td>
<td>CLINGCON</td>
</tr>
<tr>
<td></td>
<td>EZCSP</td>
</tr>
<tr>
<td></td>
<td>INCA</td>
</tr>
<tr>
<td></td>
<td>MINISAT(ID)</td>
</tr>
<tr>
<td>SMT solver</td>
<td>DINGO</td>
</tr>
<tr>
<td></td>
<td>ASPMT2SMT</td>
</tr>
<tr>
<td></td>
<td>EZSMT</td>
</tr>
<tr>
<td>Mixed integer solver</td>
<td>MINGO</td>
</tr>
</tbody>
</table>
## CASP Systems based on answer set and CSP/CLP solvers

<table>
<thead>
<tr>
<th>ACSolver</th>
<th>Answer Set Solver</th>
<th>CSP/CLP Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLINGCON</td>
<td>SMODELS CLASP</td>
<td>CLP systems</td>
</tr>
<tr>
<td>EZCSP</td>
<td>CLASP/CMODELS</td>
<td>GECODE/its own CP solver</td>
</tr>
<tr>
<td>MINISAT(ID)</td>
<td>MINISAT+unfounded propagator</td>
<td>BPROLOG/SICStus Prolog</td>
</tr>
<tr>
<td>INCA</td>
<td>CLASP</td>
<td>GECODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>its own CP solver</td>
</tr>
</tbody>
</table>
### CASP Systems based on SMT solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>SMT Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>DINGO</td>
<td>z3 (difference logic)</td>
</tr>
<tr>
<td>ASPMT2SMT</td>
<td>z3</td>
</tr>
<tr>
<td>EZSMT</td>
<td>z3 and cvc4 (via SMT-LIB)</td>
</tr>
</tbody>
</table>

### CASP System based on Mixed integer solver

<table>
<thead>
<tr>
<th>Solver</th>
<th>Mixed Integer Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINGO</td>
<td>IBM ILOG CPLEX</td>
</tr>
</tbody>
</table>
## Experimental Data

<table>
<thead>
<tr>
<th>Benchmark (# instances)</th>
<th>CLINGCON</th>
<th>EZCSP</th>
<th>EZSMT</th>
<th>EZSMT</th>
<th>MINGO</th>
<th>CMODELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cumulative Time (timeout)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rev. folding (50)</td>
<td>2014 (1)</td>
<td>559</td>
<td>47948 (22)</td>
<td>4873 (2)</td>
<td>14962 (1)</td>
<td>84616 (47)</td>
</tr>
<tr>
<td>W. Seq. (30)</td>
<td>187</td>
<td>13879</td>
<td>24.2</td>
<td>23.3</td>
<td>1330 to (30)</td>
<td></td>
</tr>
<tr>
<td>Incr. sched. (30)</td>
<td>20417 (11)</td>
<td>37332 (20)</td>
<td>10277 (5)</td>
<td>9135 (5)</td>
<td>13626 (7) to (30)</td>
<td></td>
</tr>
<tr>
<td>Job shop (100)</td>
<td>2.77</td>
<td>to (100)</td>
<td>106</td>
<td>48.8</td>
<td>1137 163106 (90)</td>
<td></td>
</tr>
<tr>
<td>Newspaper (100)</td>
<td>0.02</td>
<td>3.53</td>
<td>7.68</td>
<td>3.77</td>
<td>54.2  111615 (53)</td>
<td></td>
</tr>
<tr>
<td>Sorting (189)</td>
<td>31.7</td>
<td>103</td>
<td>646</td>
<td>233</td>
<td>8282  271004 (141)</td>
<td></td>
</tr>
</tbody>
</table>
Plan

1. Subject Matter and Motivation
2. CASP and SMT: Informally
3. Generalized Constraint Satisfaction Problem
4. Constraint Answer Set Programs and Constraint Formulas
5. SMT Theories, Restriction Formulas, and Uniform SMT Theories
6. SMT and Logic Programs Modulo Theories
7. CASP Systems Overview
8. Conclusions
Conclusions

Unifying Terminology and Relatedness

Terminology  
Unification of terminology helps relate CASP and SMT

Uniform Theories  
Readily relate SMT, GCSP, and ASPT for theories such as integer linear arithmetic and difference logic

Ontology of Solvers  
Link many solvers to uniform theories

Experimental Findings  
Are uniques as they span technology stemming from three distinct research communities.

The EZSMT Solver

This work acts as the theoretical foundations for a new CASP solver.

Conference Publications

- Constraint Answer Set Programming versus Satisfiability Modulo Theories  
  (with Ben Susman). IJCAI-2016

- SMT-based Constraint Answer Set Solver EZSMT (System Description)  
  (with Ben Susman). ICLP-2016
Thank you for your attention

Questions?