Functional Stable Model Semantics and Answer Set Programming Modulo Theories

Michael Bartholomew and Joohyung Lee
School of Computing, Informatics and Decision Systems Engineering
Arizona State University, Tempe, USA
{mjbartho, joolee}@asu.edu

Abstract

Recently there has been an increasing interest in incorporating "intensional" functions in answer set programming. Intensional functions are those whose values can be described by other functions and predicates, rather than being pre-defined as in the standard answer set programming. We demonstrate that the functional stable model semantics plays an important role in the framework of "Answer Set Programming Modulo Theories (ASPMT)" —a tight integration of answer set programming and satisfiability modulo theories, under which existing integration approaches can be viewed as special cases where the role of functions is limited. We show that "tight" ASPMT programs can be translated into SMT instances, which is similar to the known relationship between ASP and SAT.

1 Introduction

In answer set programming (ASP), variables are understood in terms of grounding, and this limits answer sets to Herbrand models, under which interpretations of functions are pre-defined: every ground term represents itself, and is distinct from each other. Nonmonotonicity of the stable model semantics is related to minimizing the extents of predicates, but has nothing to do with functions, which forces us to represent the concept of nonBoolean fluents indirectly in terms of predicates, and not by functions. A drawback of such representation is that grounding often becomes a bottleneck in computation as the value domain of a nonBoolean fluent gets large.

There are two recent groups of work to extend the stable model semantics, each of which focuses only on one aspect of the issues above. One group is interested in enriching the modeling language by incorporating "intensional" functions [Cabalar, 2011; Lifschitz, 2012; Bartholomew and Lee, 2012; Balduccini, 2012]. Intensional functions are functions whose values can be described by other functions and predicates, rather than being pre-defined. Such semantics allow us to represent nonBoolean fluents by intensional functions without having to rely on predicates.

The other group of work focuses on improving the computational efficiency by integrating ASP with other declarative computing paradigms, such as constraint processing, satisfiability modulo theories, or mixed integer programming [Gebser *et al.*, 2009; Balduccini, 2009; Janhunen *et al.*, 2011; Liu *et al.*, 2012], and exploiting the efficient constraint processing techniques on functions (or called "variables" in CSP/SMT) without having to generate a large set of ground instances. Constraint variables are functions that are mapped to values in their domains. In SMT with difference logic or linear constraints, arithmetic variables are functions that are mapped to numbers. However, these functions are not as expressive as intensional functions.

This paper combines the advantages of the two groups of work resulting in the framework of *Answer Set Programming Modulo Theories (ASPMT)*—a tight integration of answer set programming and satisfiability modulo theories, where functions are as expressive as predicates in answer set programming, and can be computed efficiently without having to ground w.r.t. their value domains. The existing languages in the second group can be viewed as special cases of this language. A fragment of ASPMT can be translated into the language of SMT, allowing SMT solvers to be used for computation.

Section 2 reviews the functional stable model semantics by Bartholomew and Lee [2012]. Section 3 presents a reformulation of this semantics in terms of grounding and reduct for infinitary ground formulas. In Section 4 we define the concept of ASPMT as a special case of the functional stable model semantics, and presents a syntactic class of ASPMT instances that can be translated into SMT instances, which in turn allows us to use SMT solvers to compute ASPMT. In Section 5 we compare ASPMT with other similar approaches, clingcon programs and ASP(LC) programs.

2 Functional Stable Model Semantics

2.1 Review of the Bartholomew-Lee Semantics

We review the stable model semantics of intensional functions from [Bartholomew and Lee, 2012]. Formulas are built the same as in first-order logic. A signature consists of *function constants* and *predicate constants*. Function constants of arity 0 are called *object constants*, and predicate constants of arity 0 are called *propositional constants*.

Similar to circumscription, for predicate symbols (constants or variables) u and c, expression $u \leq c$ is defined as shorthand for $\forall \mathbf{x}(u(\mathbf{x}) \rightarrow c(\mathbf{x}))$. Expression $u = c$ is defined as $\forall \mathbf{x}(u(\mathbf{x}) \leftrightarrow c(\mathbf{x}))$ if u and c are predicate symbols, and $\forall \mathbf{x}(u(\mathbf{x}) = c(\mathbf{x}))$ if they are function symbols. For lists of symbols $\mathbf{u} = (u_1, \dots, u_n)$ and $\mathbf{c} = (c_1, \dots, c_n)$, expression $\mathbf{u} \leq \mathbf{c}$ is defined as $(u_1 \leq c_1) \wedge \cdots \wedge (u_n \leq c_n)$, and similarly, expression $\mathbf{u} = \mathbf{c}$ is defined as $(u_1 = c_1) \wedge \cdots \wedge (u_n = c_n)$. Let $\mathbf{c}$ be a list of distinct predicate and function constants, and let $\widehat{\mathbf{c}}$ be a list of distinct predicate and function variables corresponding to $\mathbf{c}$. By $\mathbf{c}^{pred}$ ($\mathbf{c}^{func}$, respectively) we mean the list of all predicate constants (function constants, respectively) in $\mathbf{c}$, and by $\widehat{\mathbf{c}}^{pred}$ ($\widehat{\mathbf{c}}^{func}$, respectively) the list of the corresponding predicate variables (function variables, respectively) in $\widehat{\mathbf{c}}$.

For any formula F, expression $\text{SM}[F;\ \mathbf{c}]$ is defined as

$$F \wedge \neg \exists \widehat{\mathbf{c}}(\widehat{\mathbf{c}} < \mathbf{c} \wedge F^*(\widehat{\mathbf{c}})),$$

where $\widehat{\mathbf{c}} < \mathbf{c}$ is shorthand for $(\widehat{\mathbf{c}}^{pred} \leq \mathbf{c}^{pred}) \wedge \neg(\widehat{\mathbf{c}} = \mathbf{c})$, and $F^*(\widehat{\mathbf{c}})$ is defined recursively as follows.

- When F is an atomic formula, F^* is $F' \wedge F$ where F' is obtained from F by replacing all intensional (function and predicate) constants $\mathbf{c}$ in it with the corresponding (function and predicate) variables from $\widehat{\mathbf{c}}$;
- $(G \wedge H)^* = G^* \wedge H^*$; $\quad (G \vee H)^* = G^* \vee H^*$;
- $(G \rightarrow H)^* = (G^* \rightarrow H^*) \wedge (G \rightarrow H)$;
- $(\forall x G)^* = \forall x G^*$; $\quad (\exists x F)^* = \exists x F^*$.

(We understand $\neg F$ as shorthand for $F \rightarrow \bot$; $\top$ as $\neg\bot$; and $F \leftrightarrow G$ as $(F \rightarrow G) \wedge (G \rightarrow F)$.) Members of $\mathbf{c}$ are called *intensional* constants.

When F is a sentence, the models of $\text{SM}[F; \mathbf{c}]$ are called the *stable* models of F *relative to* $\mathbf{c}$. They are the models of F that are "stable" on $\mathbf{c}$. The definition can be easily extended to formulas of many-sorted signatures.

This definition of a stable model is a proper generalization of the one from [Ferraris *et al.*, 2011], which views logic programs as a special case of first-order formulas.

We will often write $G \leftarrow F$, in a rule form as in logic programs, to denote the universal closure of $F \rightarrow G$. A finite set of formulas is identified with the conjunction of the formulas in the set.

Example 1 *The following set F of formulas describes the capacity of a container of water that has a leak but that can be refilled to the maximum amount, say 10, with the action FillUp.*

$$\begin{array}{rcl} \{Amount_1 = x\} & \leftarrow & Amount_0 = x+1 \\ Amount_1 = 10 & \leftarrow & FillUp\ . \end{array}$$

Here $Amount_1$ is an intensional function constant, and x is a variable ranging over nonnegative integers. According to the semantics from [Bartholomew and Lee, 2012], the first rule is a default rule (or choice rule) standing for

$$(Amount_1 = x) \vee \neg(Amount_1 = x) \ \leftarrow\ Amount_0 = x+1, \quad (1)$$

and expresses that the amount at next time decreases by default. However, if FillUp action is executed (if we add FillUp as a fact), this behavior is overridden, and the amount is set to the maximum value.

Consider an interpretation I that has the set of nonnegative integers as the universe, interprets integers, arithmetic functions and comparison operators in the standard way, and has $FillUp^I = \text{FALSE}$, $Amount_0^I = 6$, $Amount_1^I = 5$. One can check that I is a model of $\text{SM}[F; Amount_1]$. Consider another interpretation I_1 that agrees with I except that $Amount_1^{I_1} = 8$. This is a model of F but not of $\text{SM}[F; Amount_1]$. Another interpretation I_2 that agrees with I except that $FillUp^{I_2} = \text{TRUE}$, $Amount_1^{I_2} = 10$ satisfies $\text{SM}[F; Amount_1]$.

This example demonstrates the ability to assign a default value to an intensional function, which is different from the previous value of the function.

3 FSM in terms of Grounding and Reduct

Instead of relying on a transformation into second-order logic, the definition of a stable model in the previous section can be characterized in terms of grounding and reduct. While the definition in terms of second-order logic is succinct, the reduct-based definition is more familiar, and tells us some other insights.

3.1 Review: Infinitary ground Formulas

Since the universe can be infinite, grounding a quantified sentence introduces infinite conjunctions and disjunctions over the elements in the universe. Here we rely on the concept of grounding *relative to an interpretation* from [Truszczynski, 2012]. The following is the definition of an *infinitary ground formula*, which is adapted from [Truszczynski, 2012]. The main difference between them is that we do not replace ground terms with their corresponding object names, leaving intensional functions in grounding. This is essential for defining a reduct for the functional stable model semantics.

For each element ξ of the universe $|I|$ of I, we introduce a new symbol $\xi^\diamond$, called an *object name*. By σ^I we denote the signature obtained from σ by adding all object names $\xi^\diamond$ as additional object constants. We will identify an interpretation I of signature σ with its extension to σ^I defined by $I(\xi^\diamond) = \xi$.[1]

We assume the primary connectives to be $\bot$, $\{\}^\wedge$, $\{\}^\vee$, and $\rightarrow$. Propositional connectives $\wedge, \vee, \neg, \top$ are considered as shorthands: $F \wedge G$ as $\{F, G\}^\wedge$; $F \vee G$ as $\{F, G\}^\vee$.

Let A be the set of all ground atomic formulas of signature σ. The sets $\mathcal{F}_0, \mathcal{F}_1, \dots$ are defined recursively as follows:

- $\mathcal{F}_0 = A \cup \{\bot\}$;
- $\mathcal{F}_{i+1} (i \geq 0)$ consists of expressions $\mathcal{H}^\wedge$ and $\mathcal{H}^\vee$, for all subset $\mathcal{H}$ of $\mathcal{F}_0 \cup \ldots \cup \mathcal{F}_i$, and of the expressions $F \rightarrow G$, where $F, G \in \mathcal{F}_0 \cup \cdots \cup \mathcal{F}_i$.

We define $\mathcal{L}_A^{inf} = \bigcup_{i=0}^{\infty} \mathcal{F}_i$. We call elements of $\mathcal{L}_A^{inf}$ *infinitary ground formulas* of σ.

For any interpretation I of σ and any infinitary ground formula F, the definition of satisfaction, $I \models F$, is as follows:

[1] For details, see [Lifschitz *et al.*, 2008].

- For atomic formulas, the definition is the same as in the standard first-order logic;
- $I \models \mathcal{H}^\wedge$ if for every formula $G \in \mathcal{H}$, $I \models G$;
- $I \models \mathcal{H}^\vee$ if there is a formula $G \in \mathcal{H}$ such that $I \models G$;
- $I \models G \rightarrow H$ if $I \not\models G$ or $I \models H$.

Let F be any first-order sentence of a signature σ, and let I be an interpretation of σ. By $gr_I[F]$ we denote the *infinitary ground formula w.r.t. I*, which is obtained from F by the following process:

- If F is an atomic formula, $gr_I[F]$ is F;
- $gr_I[F \odot G] = gr_I[F] \odot gr_I[G] \quad (\odot \in \{\wedge, \vee, \rightarrow\})$;
- $gr_I[\forall x F(x)] = \{gr_I[F(\xi^\diamond)] \mid \xi \in |I|\}^\wedge$;
- $gr_I[\exists x F(x)] = \{gr_I[F(\xi^\diamond)] \mid \xi \in |I|\}^\vee$.

Note that $gr_I[F]$ is an infinitary ground formula of σ^I.

Example 1 continued *Consider again F in Example 1, and the same interpretation I there. $gr_I(F)$ is the following set of formulas:*

$$\begin{array}{rcl} \{Amount_1 = 0\} & \leftarrow & Amount_0 = 0+1 \\ \{Amount_1 = 1\} & \leftarrow & Amount_0 = 1+1 \\ & \dots & \\ Amount_1 = 10 & \leftarrow & FillUp\ . \end{array} \tag{2}$$

3.2 Reduct-Based Definition of SM

For any two interpretations I, J of the same signature and any list $\mathbf{c}$ of distinct predicate and function constants, we write $J <^{\mathbf{c}} I$ if

- J and I have the same universe and agree on all constants not in $\mathbf{c}$;
- $p^J \subseteq p^I$ for all predicate constants p in $\mathbf{c}$; and
- J and I do not agree on $\mathbf{c}$.

The *reduct* $F^{\underline{I}}$ of an infinitary ground formula F relative to an interpretation I is defined as follows:

- For each atomic formula F, $F^{\underline{I}} = \bot$ if $I \not\models F$ and $F^{\underline{I}} = F$ otherwise;
- $(\mathcal{H}^\wedge)^{\underline{I}} = \bot$ if $I \not\models \mathcal{H}^\wedge$; otherwise $(\mathcal{H}^\wedge)^{\underline{I}} = \{G^{\underline{I}} \mid G \in \mathcal{H}\}^\wedge$;
- $(\mathcal{H}^\vee)^{\underline{I}} = \bot$ if $I \not\models \mathcal{H}^\vee$; otherwise $(\mathcal{H}^\vee)^{\underline{I}} = \{G^{\underline{I}} \mid G \in \mathcal{H}\}^\vee$;
- $(G \rightarrow H)^{\underline{I}} = \bot$ if $I \not\models G \rightarrow H$; otherwise $(G \rightarrow H)^{\underline{I}} = G^{\underline{I}} \rightarrow H^{\underline{I}}$.

The following theorem states the reformulation of FSM in terms of grounding and reduct.

Theorem 1 *Let F be a first-order sentence and $\mathbf{c}$ a list of intensional constants. For any interpretation I of σ, $I \models$ SM$[F; \mathbf{c}]$ iff*

- *I satisfies F, and*
- *every interpretation J such that $J <^{\mathbf{c}} I$ does not satisfy $(gr_I[F])^{\underline{I}}$.*

Example 1 continued *Among the ground formulas in* (2)*, only the implication*

$$\{Amount_1 = 5\} \ \leftarrow \ Amount_0 = 5+1$$

which stands for

$$(Amount_1 = 5) \vee \neg(Amount_1 = 5) \ \leftarrow \ Amount_0 = 5+1$$

has its antecedent satisfied by I, so the reduct $(gr_I[F])^{\underline{I}}$ is equivalent to

$$(Amount_1 = 5) \vee \bot \ \leftarrow \ Amount_0 = 5+1$$

No interpretation that is different from I only on $Amount_1$ satisfies the reduct. On the other hand, the reduct $(gr_{I_1}[F])^{\underline{I_1}}$ is equivalent to

$$\bot \vee \neg\bot \ \leftarrow \ Amount_0 = 5+1,$$

and other interpretations that are different from I_1 only on $Amount_1$ satisfy the reduct. $(gr_{I_2}[F])^{\underline{I_2}}$ is equivalent to

$$\begin{array}{l} \bot \vee \neg\bot \ \leftarrow \ Amount_0 = 5+1, \\ Amount_1 = 10 \ \leftarrow \ FillUp \end{array}$$

and I_2 is the only interpretation that satisfies the reduct.

In accordance with Theorem 1, I and I_2 are the stable models of F relative to $Amount_1$, but I_1 is not.

4 Answer Set Programming Modulo Theories

4.1 ASPMT as a Special Case of FSM

Formally, an SMT instance is a formula in many-sorted first-order logic, where some designated function and predicate constants are constrained by some fixed background interpretation. SMT is the problem of determining whether such a formula has a model that expands the background interpretation [Barrett *et al.*, 2009].

The syntax of ASPMT is the same as that of SMT. Let σ^{bg} be the (many-sorted) signature of the background theory bg. An interpretation of σ^{bg} is called a *background interpretation* if it satisfies the background theory. For instance, in the theory of reals, we assume that σ^{bg} contains the set $\mathcal{R}$ of symbols for all real numbers, the set of arithmetic functions over real numbers, and the set $\{<, >, \leq, \geq\}$ of binary predicates over real numbers. Background interpretations interpret these symbols in the standard way.

Let σ be a signature that is disjoint from σ^{bg}. We say that an interpretation I of σ satisfies F w.r.t. the background theory bg, denoted by $I \models_{bg} F$, if there is a background interpretation J of σ^{bg} that has the same universe as I, and $I \cup J$ satisfies F. For any ASPMT sentence F with background theory σ^{bg}, interpretation I is a stable model of F relative to $\mathbf{c}$ (w.r.t. background theory σ^{bg}) if $I \models_{bg}$ SM$[F; \mathbf{c}]$.

Example 1 continued *Formula F can be understood as an ASPMT formula with the theory of integers as the background theory. Arithmetic functions and comparison operators belong to the background signature. If I' is an interpretation of signature $\{Amount_0, Amount_1, FillUp\}$ which agrees with I on these constants, We say that $I' \models_{bg}$ SM$[F; Amount_1]$.*

Instance Size	CLINGO v3.0.5 Execution Run Time (Grounding + Solving)	Atoms	iSAT v1.0 Execution Run Time (Last step)	Variables	Z3 V4.3.0 Execution Run time	Memory
10	0s (0s + 0s)	210	0s(0s)	86	.05s	2.07
50	.02s (.02s + 0s)	2970	.05s(0s)	406	.18s	2.17
100	.12s (.12s + 0s)	10920	.15s(0s)	806	.33s	2.28
500	8.18s (8.17s + 0.01s)	254520	4.41s(.03s)	4006	1.68	3.38s
1000	55.17s (55.15s + 0.02s)	1009020	18.57s(.09s)	8006	3.35s	4.73
5000	Did not terminate in 2 hours		500.17s(.45s)	40006	17.42s	17.32
10000	Did not terminate in 2 hours		2008.97s(.93s)	80006	36.49s	31.42

Table 1: Leaking Bucket Experiment Results

4.2 Turning ASPMT into SMT for Tight Programs

We say that a formula F is in *Clark normal form* (relative to the list $\mathbf{c}$ of intensional constants) if it is a conjunction of sentences of the form

$$\forall \mathbf{x}(G \rightarrow p(\mathbf{x})) \qquad (3)$$

and

$$\forall \mathbf{x}y(G \rightarrow f(\mathbf{x})=y) \qquad (4)$$

one for each intensional predicate p and each intensional function f, where $\mathbf{x}$ is a list of distinct object variables, y is an object variable, and G is an arbitrary formula that has no free variables other than those in $\mathbf{x}$ and y.

The *completion* of a formula F in Clark normal form (relative to $\mathbf{c}$) is obtained from F by replacing each conjunctive term (3) with

$$\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow G)$$

and each conjunctive term (4) with

$$\forall \mathbf{x}y(f(\mathbf{x})=y \leftrightarrow G).$$

An occurrence of a symbol or a subformula in a formula F is called *strictly positive* in F if that occurrence is not in the antecedent of any implication in F. The *t-dependency graph* of F (relative to $\mathbf{c}$) is the directed graph that

- has all members of $\mathbf{c}$ as its vertices, and
- has an edge from c to d if, for some strictly positive occurrence of $G \rightarrow H$ in F,
  - c has a strictly positive occurrence in H, and
  - d has a strictly positive occurrence in G.

We say that F is *tight* (on $\mathbf{c}$) if the t-dependency graph of F (relative to $\mathbf{c}$) is acyclic. For example,

$$((p \rightarrow q) \rightarrow r) \rightarrow p$$

is tight on $\{p, q, r\}$ because its t-dependency graph has only one edge, which goes from p to r. On the other hand, the formula is not tight according to [Ferraris *et al.*, 2011] because, according to the definition of a dependency graph in that paper, there is an additional edge that goes from p to itself.

Theorem 12 from [Bartholomew and Lee, 2012] extended the theorem on completion from [Ferraris *et al.*, 2011] to allow intensional functions, but it was restricted to a class of formulas called $\mathbf{c}$-plain formulas. The following theorem generalizes that theorem by removing that restriction and by referring to the weaker notion of tightness as described above.

Theorem 2 *For any sentence F in Clark normal form that is tight on $\mathbf{c}$, an interpretation I that satisfies $\exists xy(x \neq y)$ is a model of* SM$[F; \mathbf{c}]$ *iff I is a model of the completion of F relative to $\mathbf{c}$.*

Example 1 continued *Formula (1) is strongly equivalent to*

$$Amount_1 = x \leftarrow \neg\neg(Amount_1 = x) \wedge Amount_0 = x+1,$$

so that formula F in Example 1 can be turned into Clark normal form relative to $Amount_1$:

$$Amount_1 = x \ \leftarrow \ (\neg\neg(Amount_1 = x) \wedge Amount_0 = x+1) \\ \vee\ (x = 10 \wedge FillUp).$$

and the completion turns it into

$$Amount_1 = x \ \leftrightarrow \ (\neg\neg(Amount_1 = x) \wedge Amount_0 = x+1) \\ \vee\ (x = 10 \wedge FillUp).$$

Using equality, the formula can be written without mentioning the variable x as

$$(Amount_0 = Amount_1 + 1) \vee (Amount_1 = 10 \wedge FillUp) \\ FillUp \rightarrow Amount_1 = 10\ .$$

In the language of iSAT, this formula can be represented as

```
(Amt = Amt'+1) or (Amt'=10 and FillUp);
FillUp -> Amt'=10;
```

and in the language of Z3, it can be represented as

```
(assert (or (= Amt0 (+ Amt1 1))
  (and (= Amt1 10) FillUp)))
(assert (=> FillUp0 (= Amt1 10))) .
```

Alternatively, according to the method in [Bartholomew and Lee, 2012], formula F in Example 1 can be turned into the input language of GRINGO by eliminating intensional functions in favor of intensional predicates.

Our first experiment has the bucket initially at capacity 5 and the goal is to get the bucket to capacity 10 at a certain fixed timepoint. The different instance sizes correspond to the maximum capacity of the bucket and the certain timepoint (they are both the same in each case). iSAT finds a model of bounded length k, where k starts from 0 and increases by 1 until a model is found. The run time reported is the total cumulative times for $k = 0, 1, \dots, m$ where m is the instance size. The last step time is for the run when $k = m$. For other systems, we fixed the length $k = m$ from the beginning. The results shown in Table 1 demonstrate that even for a relatively simple domain, ASP suffers a grounding bottleneck that is not present when using SMT solvers. We see that the number of

Instance Size	CLINGO v3.0.5 Execution		iSAT v1.0 Execution		Z3 V4.3.0 Execution	
	Run Time (Grounding + Solving)	Atoms	Run Time	Variables	Run time	Memory
5	.02s (.02s + 0s)	3174	.03s	331	.03s	2.79
10	.3s (.3s + 0s)	10161	.19s	596	.09s	4.91
20	9.46s (4.02s + 5.11s)	36695	.79s	1126	.2s	8.65
30	42.56s (22.32s + 20.24s)	77627	2.05s	1656	.36s	12.22
50	923.74s (297.26 + 626.48s)	207706	14.35s	2716	1.09s	20.35
100	out of memory		494.77s	5366	5.52s	43.86

Table 2: Gears World Experiment Results

atoms for CLINGO increases quadratically to the instance size while the number of variables for iSAT increases linearly.

Next, consider the Gears World domain in which we have two gears, Gear1 with radius 7 and Gear2 with radius 17. Each gear is connected to a motor that has integral running speeds which can be incremented by 1 using the corresponding action. The gears can also be moved close together so that both gears spin at the speed of the higher value (between *M1Speed* $\times$ *Radius1* or *M2Speed* $\times$ *Radius2*). The goal is to have Gear1 spinning at a multiple of Gear2's radius. That multiple is the instance size in Table 2, so for example, instance size 3 means at the end, we want Gear1 spinning at a speed of $51 (= 3 \times 17)$. This domain can be expressed in the language of ASPMT with the theory of integers. Here is a part of the program that governs the speed of a motor. The speed does not change unless the increase action happens, in which case it is incremented by 1 unit.

$$\begin{array}{lll} \mathit{M1Speed}_t = x & \leftarrow & \mathit{M1Speed}_{t-1} = x-1 \wedge \mathit{IncreaseM1}_{t-1} \\ \mathit{M1Speed}_t = x & \leftarrow & \neg\neg(\mathit{M1Speed}_t = x) \wedge \mathit{M1Speed}_{t-1} = x \end{array}$$

(t is a step counter, which can be represented by an ASP variable to be grounded).

The ASPMT description of the Gears World is tight and can be turned into the input language of SMT solvers by completion. For example, the completion relative to $\mathit{M1Speed}_t$ is

$$\begin{array}{l} \mathit{M1Speeed}_t = x \leftrightarrow (\mathit{M1Speed}_{t-1} = x - 1 \wedge \mathit{IncreaseM1}_{t-1}) \\ \qquad \vee(\mathit{M1Speed}_{t-1} = x \wedge \neg\neg \mathit{M1Speed}_t = x)\ . \end{array}$$

The Gears World domain can also be computed by ASP solvers by eliminating the intensional functions in favor of intensional predicates as described in [Bartholomew and Lee, 2012]. Table 2 compares the two approaches and reveals that the SMT solvers iSAT and Z3 were able to perform comparatively very well as the instance size increased.

The experiments were performed on an Intel Core 2 Duo CPU 3.00 GHz with 4 GB RAM.

5 Comparison with Other Approaches to ASP Modulo Theories

5.1 Clingcon programs as a special case of ASPMT

A *constraint satisfaction problem* (CSP) is a tuple (V, D, C), where V is a set of *constraint variables* with the respective *domains D*, and C is a set of *constraints* that specify legal assignments of values in the domains to the constraint variables.

A *clingcon program* Π with a constraint satisfaction problem (V, D, C) is a set of rules of the form

$$a \leftarrow B, N, \mathit{Cn}, \tag{5}$$

where a is a propositional atom or $\bot$, B is a set of positive propositional literals, N is a set of negative propositional literals, and *Cn* is a set of constraints from C, possibly preceded by *not*.

Clingcon programs can be viewed as ASPMT instances. Below is a reformulation of the semantics in terms of ASPMT. We assume that constraints are expressed by ASPMT sentences of signature $V \cup \sigma^{bg}$, where V is a set of object constants identified with constraint variables V in (V, D, C), whose value sorts are identified with domains in D; we assume that σ^{bg} is disjoint from V and contains all values in D as object constants, and other symbols to represent constraints, such as $+$, $\times$, and $\geq$. In other words, we represent a constraint as a formula $F(v_1, \dots, v_n)$ over $V \cup \sigma^{bg}$ where $F(x_1, \dots, x_n)$ is a formula of the signature σ^{bg} and $F(v_1, \dots, v_n)$ is obtained from $F(x_1, \dots, x_n)$ by substituting the object constants $(v_1, \dots, v_n)$ in V for $(x_1, \dots, x_n)$.

For any signature σ that consists of object constants and propositional constants, we identify an interpretation I of σ as the tuple $\langle I^f, X\rangle$, where I^f is the restriction of I on the object constants in σ, and X is a set of propositional constants in σ that are true under I.

Given a clingcon program Π with (V, D, C), and an interpretation $I = \langle I^f, X\rangle$, we define the *constraint reduct of* Π *relative to* X *and* I^f (denoted by $\Pi^X_{I^f}$) as the set of rules $a \leftarrow B$ for each rule (5) is in Π such that $I^f \models_{bg} \mathit{Cn}$, and $X \models N$. We say that a set X of propositional atoms is a *constraint answer set* of Π relative to I^f if X is a minimal model of $\Pi^X_{I^f}$.

Example 1 continued *The rules*

$$\begin{array}{l} \mathit{Amount}_1 + 1 =^{\$} \mathit{Amount}_0 \leftarrow \mathit{not\ FillUp}, \\ \mathit{Amount}_1 =^{\$} 10 \leftarrow \mathit{FillUp} \end{array}$$

are identified with

$$\begin{array}{l} \bot \leftarrow \mathit{not\ FillUp}, \mathit{not}(\mathit{Amount}_1 + 1 =^{\$} \mathit{Amount}_0) \\ \bot \leftarrow \mathit{FillUp}, \mathit{not}(\mathit{Amount}_1 =^{\$} 10) \end{array}$$

under the semantics of clingcon programs. Consider I in Example 1, which can be represented as $\langle I^f, X\rangle$ where I^f maps Amount_0 to 6, and Amount_1 to 5, and $X = \emptyset$. X is the constraint answer set relative to I^f because X is the minimal model of the constraint reduct relative to X and I^f, which is the empty set.

Similar to the way that rules are identified as a special case of formulas [Ferraris *et al.*, 2011], we identify a clingcon program Π with the conjunction of implications $B \wedge N \wedge Cn \rightarrow a$ for all rules (5) in Π. The following theorem tells us that clingcon programs are a special case of ASPMT, in which the background theory is specified by (V, D, C), and intensional constants are limited to propositional constants only, and do not allow function constants.

Theorem 3 *Let Π be a clingcon program with CSP (V, D, C), let $\mathbf{p}$ be the set of all propositional constants occurring in Π, and let I be an interpretation $\langle I^f, X\rangle$ of signature $V \cup \mathbf{p}$. Set X is a constraint answer set of Π relative to I^f iff $I \models_{bg}$ SM$[\Pi; \mathbf{p}]$.*

Note that a clingcon program does not allow an atom that consists of elements from both V and $\mathbf{p}$. Thus the truth value of any atom is determined by either I^f or X, but not by involving both of them. This allows loose coupling of an ASP solver and a constraint solver. On the other hand, [Gebser *et al.*, 2009] sketches a method to extend clingcon programs to allow predicate constants of positive arity, possibly containing constraint variables as arguments. This however leads to some unintuitive cases under the semantics of CLINGCON programs, as the following example shows.

```
$domain(100..199).  % Office numbers
myoffice(a).        % a is my office number,
:- myoffice(b).     % and b is not.
:- not a $==b.      % Nevertheless, a equals b.
```

System CLINGCON does not notice that this set of assumptions is inconsistent. This is because symbols a and b in ASP atoms and the same symbols in the constraint are not related. On the other hand, ASPMT, which allows first-order signatures, does not have this anomaly; there is no stable model under ASPMT.

5.2 Comparison with ASP(LC) Programs by Liu *et al.*

[Liu *et al.*, 2012] considers logic programs with linear constraints, or *ASP(LC)* programs, comprised of rules of the form

$$a \leftarrow B, N, LC \qquad (6)$$

where a is a propositional atom or $\bot$, B is a set of positive propositional literals, and N is a set of negative propositional literals, and LC is a set of *theory atoms*—linear constraints of the form $\sum_{i=1}^{n}(c_i \times x_i) \ \bowtie\ k$ where $\bowtie \in \{\leq, \geq, =\}$, each x_i is an object constant whose value sort is integers (or reals), and each c_i, k is an integer (or real).

An ASP(LC) program Π can be viewed as an ASPMT formula whose background theory bg is the theory of integers or the theory of reals. Let σ^p denote the set of all propositional atoms occurring in Π and σ^f denote all object constants occurring in Π that do not belong to the background signature. Theory atoms are essentially ASPMT formulas of signature $\sigma^f \cup \sigma^{bg}$. We identify ASP(LC) program Π with the conjunction of ASPMT formulas $B \wedge N \wedge LC \rightarrow a$ for all rules (6) in Π.

An *LJN-intepretation* is a pair (X, T) where $X \subseteq \sigma^p$ and T is a subset of theory atoms occurring in Π such that there is some interpretation I of signature σ^f such that $I \models_{bg} T \cup \overline{T}$, where $\overline{T}$ is the set of negations of each theory atom occurring in Π but not in T. An LJN-interpretation (X, T) satisfies an atom b if $b \in X$, the negation of an atom $not\ c$ if $c \notin X$, and a theory atom t if $t \in T$. The notion of satisfaction is extended to other propositional connectives as usual.

The *LJN-reduct* of a program Π with respect to an LJN-interpretation (X, T), denoted by $\Pi^{(X,T)}$, consists of rules $a \leftarrow B$ for each rule (6) such that (X, T) satisfies $N \wedge LC$. (X, T) is an *LJN-answer set* of Π if (X, T) satisfies Π, and X is the smallest set of atoms satisfying $\Pi^{(X,T)}$.

The following theorem tells us that there is a one-to-many relationship between LJN-answer sets and the stable models in the sense of ASPMT.

Theorem 4 *Let Π be an ASP(LC) program, and σ^p and σ^f are defined as above.*

(a) If (X, T) is an LJN-answer set of Π, then for any interpretation $\langle I^f, X\rangle$ of signature $\sigma^p \cup \sigma^f$ such that $I^f \models_{bg} T \cup \overline{T}$, we have $\langle I^f, X\rangle \models_{bg}$ SM$[\Pi; \sigma^p]$.

(b) For any interpretation $I = \langle I^f, X\rangle$ of signature $\sigma^p \cup \sigma^f$, if $\langle I^f, X\rangle \models_{bg}$ SM$[\Pi; \sigma^p]$, then an LJN-interpretation (X, T) where

$$T = \{t \mid t \text{ is a theory atom in } \Pi \text{ such that } I^f \models_{bg} t\}$$

is an LJN-answer set of Π.

Example 2 *Let F be*

$$\begin{array}{ll} a \leftarrow x - z > 0. & b \leftarrow x - y \leq 0. \\ c \leftarrow b,\ y - z \leq 0. & \leftarrow not\ a. \\ b \leftarrow c. & \end{array}$$

The LJN-interpretation $L = \langle\{a\}, \{x - z > 0\}\rangle$ is an answer set of F since $\{(x - z > 0, \neg(x - y \leq 0), \neg(y - z \leq 0)\}$ is satisfiable (e.g. take $x^I = 2, y^I = 1, z^I = 0$) and the set $\{a\}$ is the minimal model satisfying the reduct $F^L = (\top \rightarrow a) \wedge c \rightarrow b$. On the other hand the interpretation I such that $x^I = 2, y^I = 1, z^I = 0, a^I = \text{TRUE}, b^I = \text{FALSE}, c^I = \text{FALSE}$ satisfies $I \models_{bg}$ SM$[F; abc]$.

As with clingcon programs, ASP(LC) programs are more restrictive than ASPMT. ASP(LC) programs do not allow theory atoms in the head of a rule, and like clingcon programs, cannot express intensional functions.

6 Conclusion

In this paper, we related the two lines of research on functions in answer set programming that originated from different motivations, leading to an expressive KR formalism called ASPMT, which can be efficiently computed by SMT solvers. The relationship between ASPMT and SMT is similar to the relationship between ASP and SAT. We expect that, in addition to completion, many results known between ASP and SAT can be carried over to the relationship between ASPMT and SMT.

Acknowledgements

We are grateful to Yunsong Meng for useful discussions related to this paper. We are also grateful to the anonymous referees for their useful comments. This work was partially supported by the National Science Foundation under Grant IIS-0916116 and by the South Korea IT R&D program MKE/KIAT 2010-TD-300404-001.

References

[Balduccini, 2009] Marcello Balduccini. Representing constraint satisfaction problems in answer set programming. In *Working Notes of the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*, 2009.

[Balduccini, 2012] Marcello Balduccini. A ”conservative” approach to extending answer set programming with non-herbrand functions. In *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, pages 24–39, 2012.

[Barrett *et al.*, 2009] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[Bartholomew and Lee, 2012] Michael Bartholomew and Joohyung Lee. Stable models of formulas with intensional functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 2–12, 2012.

[Cabalar, 2011] Pedro Cabalar. Functional answer set programming. *TPLP*, 11(2-3):203–233, 2011.

[Ferraris *et al.*, 2011] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription. *Artificial Intelligence*, 175:236–263, 2011.

[Gebser *et al.*, 2009] M. Gebser, M. Ostrowski, and T. Schaub. Constraint answer set solving. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 235–249, 2009.

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

[Lifschitz *et al.*, 2008] Vladimir Lifschitz, Leora Morgenstern, and David Plaisted. Knowledge representation and classical logic. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, pages 3–88. Elsevier, 2008.

[Lifschitz, 2012] Vladimir Lifschitz. Logic programs with intensional functions. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 24–31, 2012.

[Liu *et al.*, 2012] Guohua Liu, Tomi Janhunen, and Ilkka Niemelä. Answer set programming via mixed integer programming. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 32–42, 2012.

[Truszczynski, 2012] Miroslaw Truszczynski. Connecting first-order ASP and the logic FO(ID) through reducts. In *Correct Reasoning*, pages 543–559, 2012.