Answer Set Programming Modulo Theories

Joohyung Lee

Automated Reasoning Group
Arizona State University, USA

AAAI 2016 Tutorial

The slides are available at

`http://peace.eas.asu.edu/aaai16tutorial`

Joint work with Michael Bartholomew, Joseph Babb, and Nikhil Loney

The project was sponsored by NSF IIS-1319794

Abstract

Answer set programming (ASP) is a successful declarative programming method oriented towards solving combinatorial and knowledge intensive problems. It has well-developed foundations, efficient reasoning systems, and a methodology of use tested on a number of industrial applications. The relationship between ASP and propositional satisfiability (SAT) has led to a method of computing answer sets using SAT solvers and techniques adapted from SAT.

Some recent extensions of ASP are to overcome the propositional setting of ASP by extending its mathematical foundation and integrating ASP with other computing paradigms. The tutorial will cover Answer Set Programming Modulo Theories, which tightly integrates ASP with Satisfiability Modulo Theories (SMT), thereby overcoming some of the computational limitations of ASP and some of the modeling limitations of SMT. A high level action language based on ASPMT allows for succinct representation of hybrid transition systems, where discrete changes and continuous changes coexist. In a broader sense, ASPMT covers an extension of ASP combined with any external computation sources.

General Messages

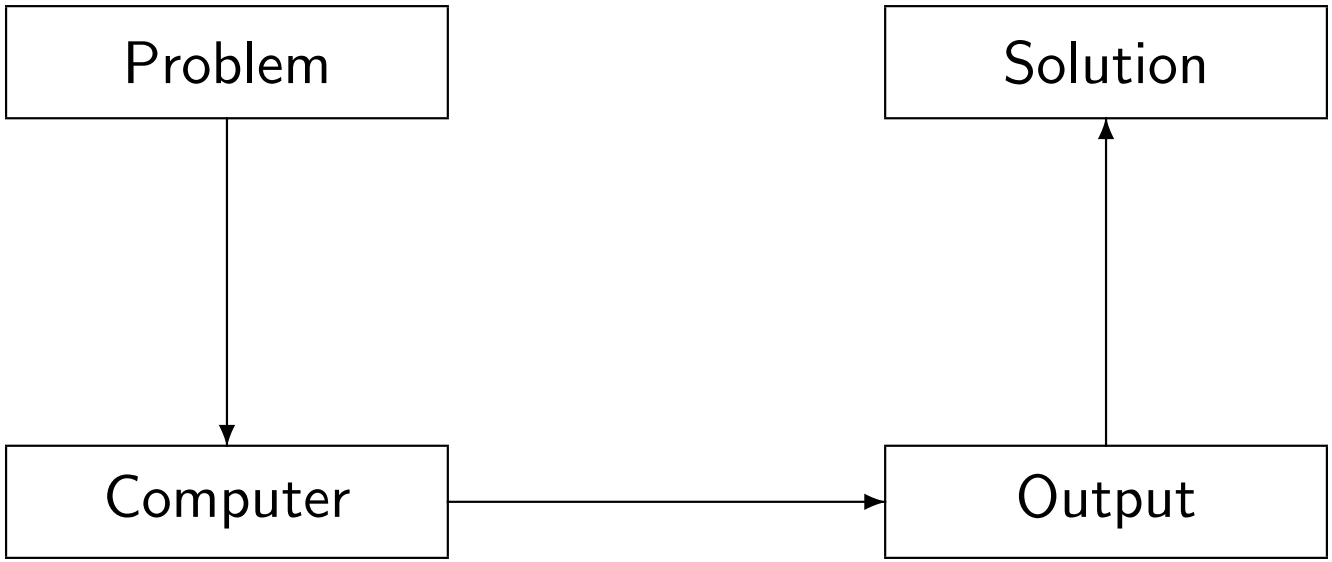
- ASP provides an elegant knowledge specification language,
  - allowing for various high level knowledge to be represented, while
  - computation can be carried out by different solvers/engines.
- First-order stable model semantics, taking into account default functions, provides a solid foundation for integrating ASP with other declarative paradigms.
- It also presents a simpler representation method in comparison with traditional ASP.
- In particular, high level action languages can be defined based on it in a simpler way.

Contents

Introduction

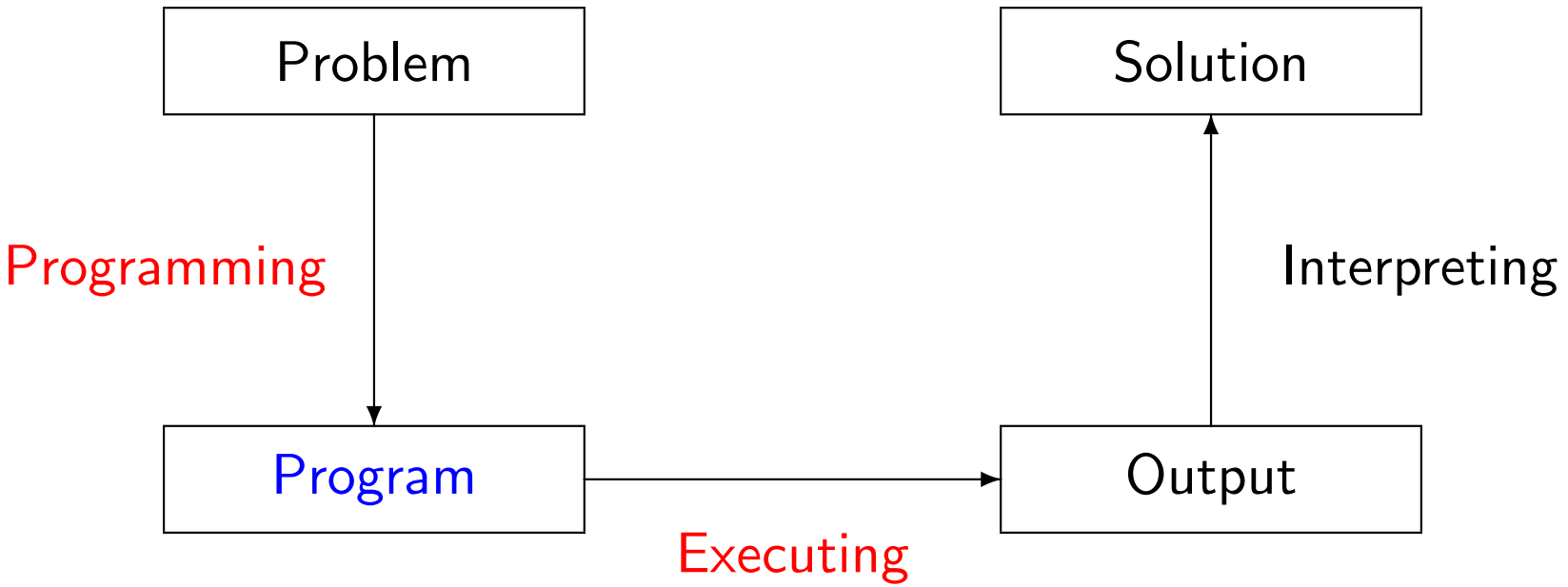
Problem Solving

"What is the problem?" versus "How to solve the problem?"



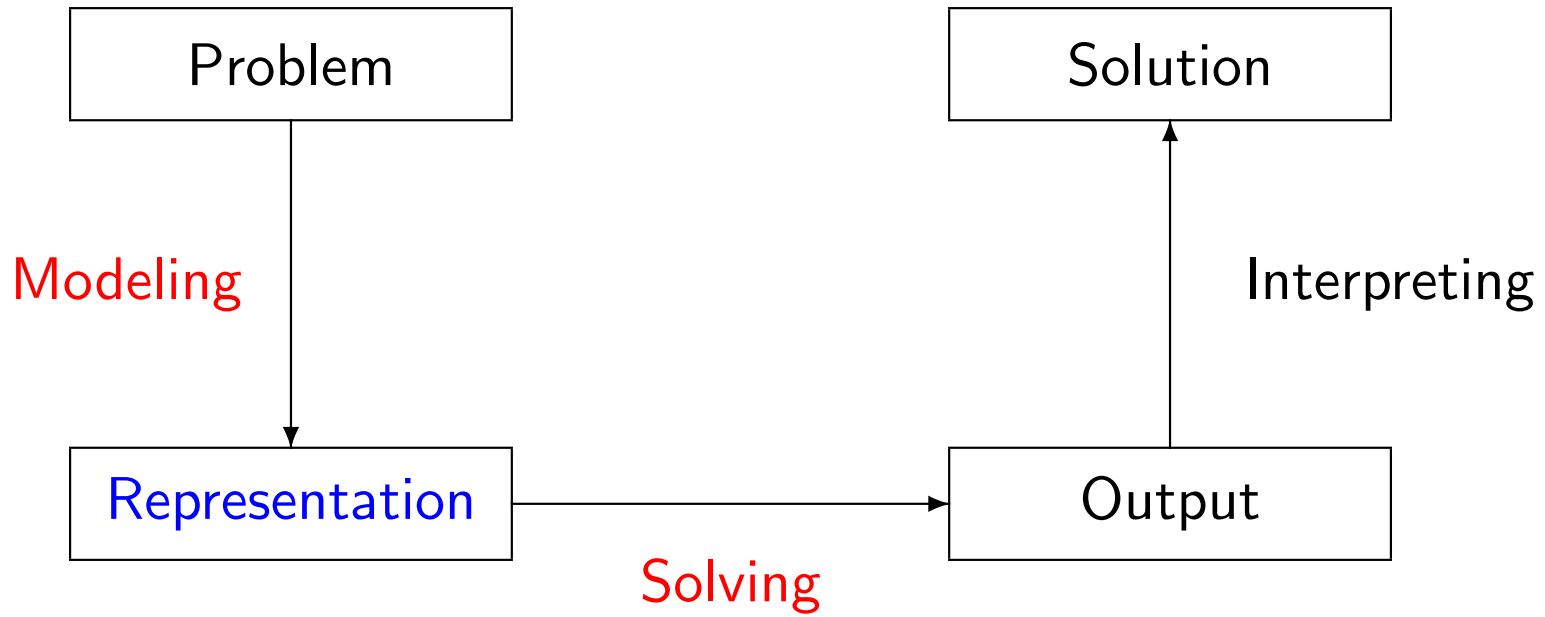
Traditional Programming

"What is the problem?" versus "How to solve the problem?"



Declarative Problem Solving

"What is the problem?" versus "How to solve the problem?"



What is Answer Set Programming (ASP)

- Declarative programming paradigm suitable for knowledge intensive and combinatorial search problems.
- Theoretical basis: answer set semantics [GL88].
- Expressive representation language: defaults, recursive definitions, aggregates, preferences, etc.
- ASP has roots in
  - deductive database
  - logic programming
  - knowledge representation
  - constraint solving (in particular SAT)

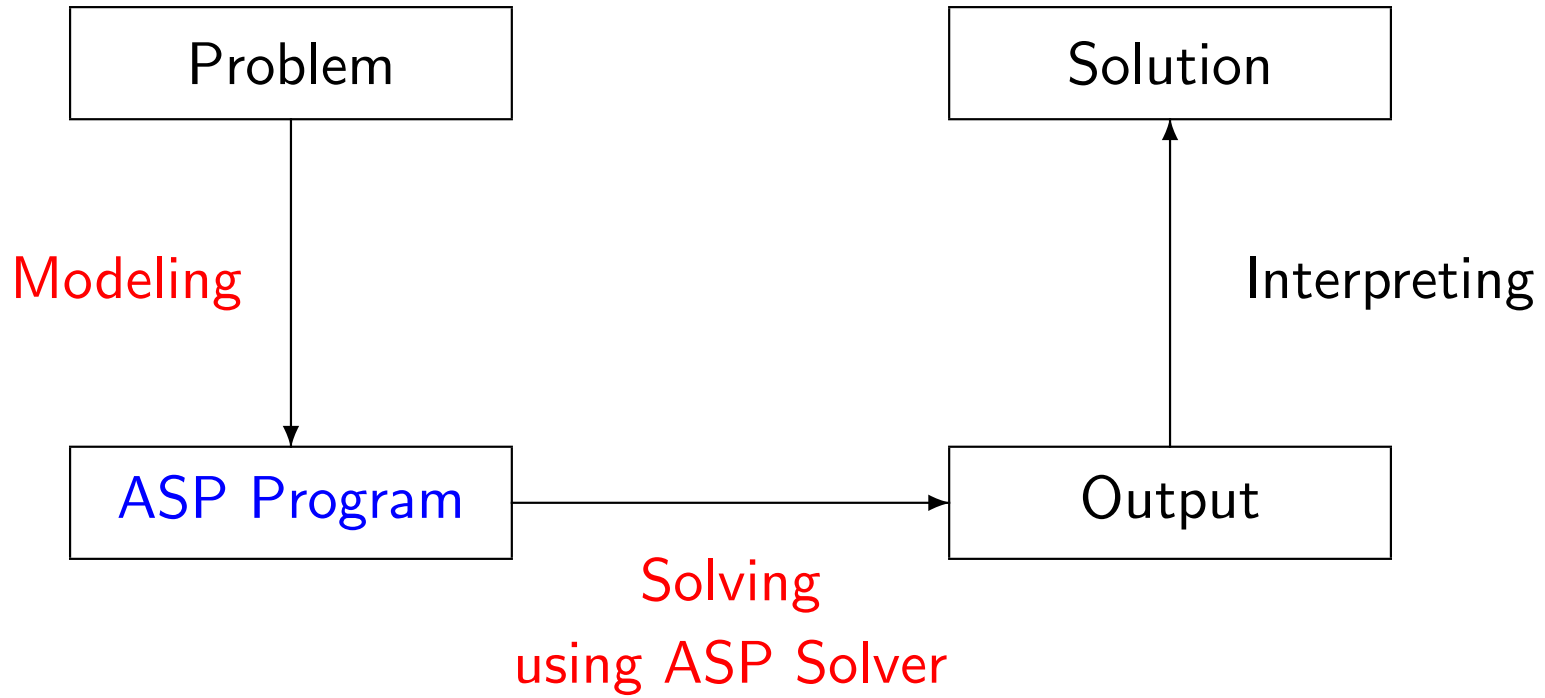
What is Answer Set Programming (ASP)

- ASP solvers:
  - SMODELS (Helsinki University of Technology, 1996)
  - DLV (Vienna University of Technology, 1997)
  - CMODELS (University of Texas at Austin, 2002)
  - PBMODELS (University of Kentucky, 2005)
  - CLASP (University of Potsdam, 2006) – winning several first places at ASP, SAT, Max-SAT, PB, CADE competitions
  - DLV-HEX for computing HEX programs.
  - OCLINGO for reactive answer set programming.
- ASP Core 2: standard language
- Annual ASP Competition

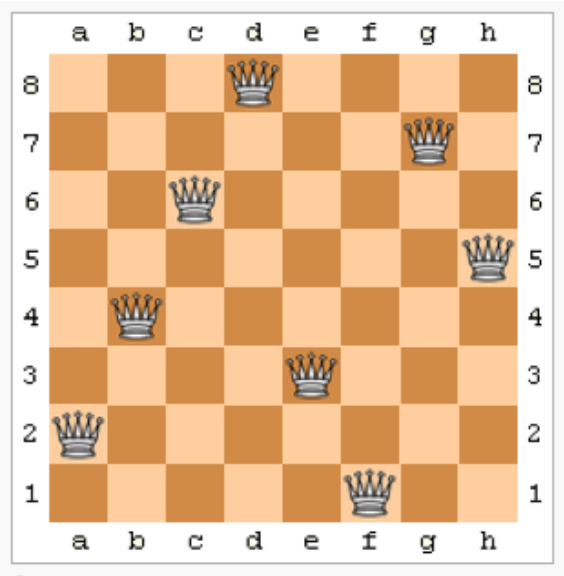
Declarative Problem Solving using ASP

The basic idea is

- to represent the given problem by a set of rules,
- to find answer sets for the program using an ASP solver, and
- to extract the solutions from the answer sets.



N-Queens Puzzle in the Language of CLINGO




```
num(1..n).

% Each column has exactly one queen
1{q(I,J) : num(I)}1 :- num(J).

% Two queens cannot stay on the same row
:- q(I,J), q(I,J1), J<J1.

% Two queens cannot stay on the same diagonal
:- q(I,J), q(I1,J1), J<J1, |I1-I|==J1-J.
```

Finding One solution for the 8-Queens Problem

With command line

```
% clingo queens -c n=8
```

The output:

```
Answer: 1
q(4,1) q(6,2) q(8,3) q(2,4) q(7,5) q(1,6) q(3,7) q(5,8)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.003s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Finding All Solutions for the 8-Queens Problem

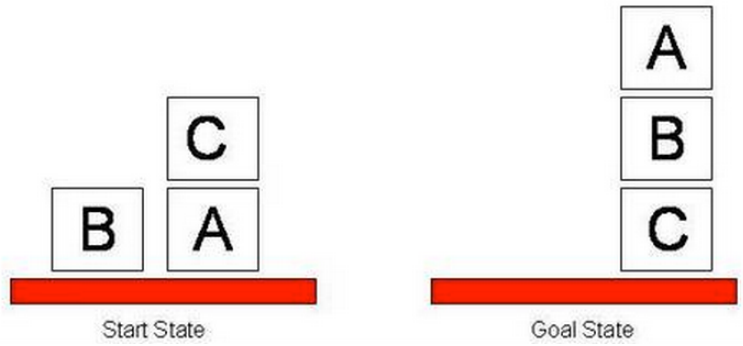
With the command line

```
% clingo queens -c n=8 0
```

CLINGO computes and shows all 92 valid queen arrangements. For instance, the last one is

```
Answer: 92
q(1,2) q(5,1) q(8,3) q(4,4) q(2,5) q(7,6) q(3,7) q(6,8)
SATISFIABLE

Models       : 92
Calls        : 1
Time         : 0.009s (Solving: 0.01s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Answer Set Planning [Lif02]



Encode a planning problem as a logic program whose answer sets correspond to solutions. Run ASP solvers to find the solutions.

Can be viewed as enhanced SAT planning [KS92].

- presents an elegant solution to the frame problem

```
on(B,L,T+1) :- on(B,L,T), not -on(B,L,T+1).
```

- indirect effects

```
above(B,L,T) :- on(B,L,T).
above(B,L,T) :- on(B,B1,T), above(B1,L,T).
```

- defeasible rules

Applications of ASP

- information integration
- constraint satisfaction
- planning, routing
- robotics
- diagnosis
- security analysis
- configuration
- computer-aided verification
- biology / biomedicine
- knowledge management
- ...

What Led to the Success of ASP?

- A simple, mathematically elegant semantics, based on the concept of a stable model
  - nonmonotonic reasoning, causal reasoning, commonsense reasoning
- Intelligent grounding—the process that replaces first-order variables with corresponding ground instances
- Efficient search methods that originated from propositional satisfiability solvers (SAT solvers)

Various Extensions

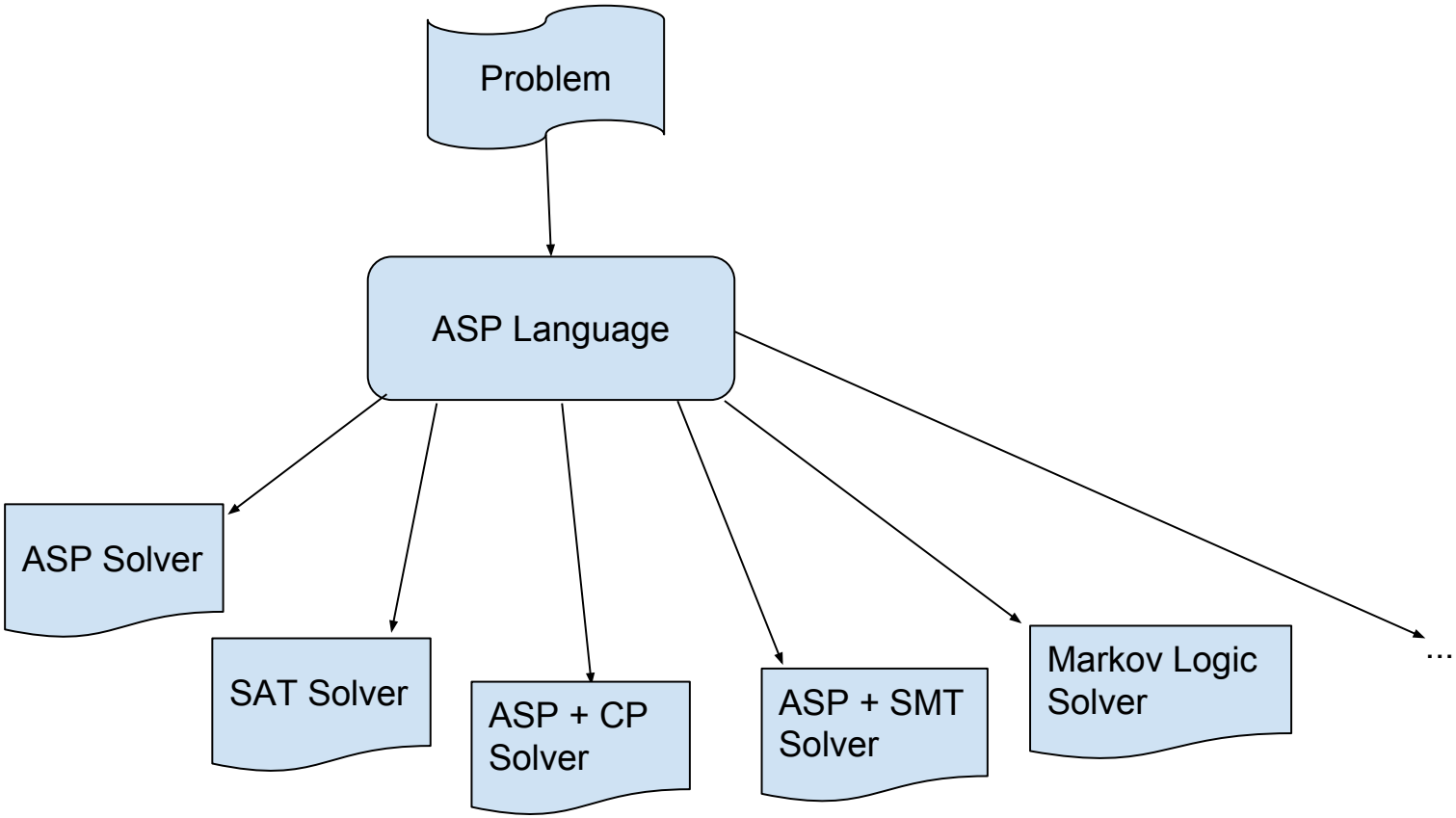
Starting from the Prolog syntax, the language of ASP has evolved:

- Strong negation [GLR91]
- Choice rules [SNS02]
- Aggregates [SNS02, FLP04, Fer05, PDB07, LM09, FL10], ...
- Preferences [BNT08]
- Integration with CSP [Bal09, GOS09]
- Integration with SMT [JLN11]
- Integration with Description Logics [EIL$^+$08, LP11]
- Integration with fuzzy logics [Luk06, LW14]
- Probabilistic answer sets [BGR09]
- Markov Logic style weighted rules [LW16]
- ...

ASP as an Interface Language

ASP language serves as a specification language for AI.

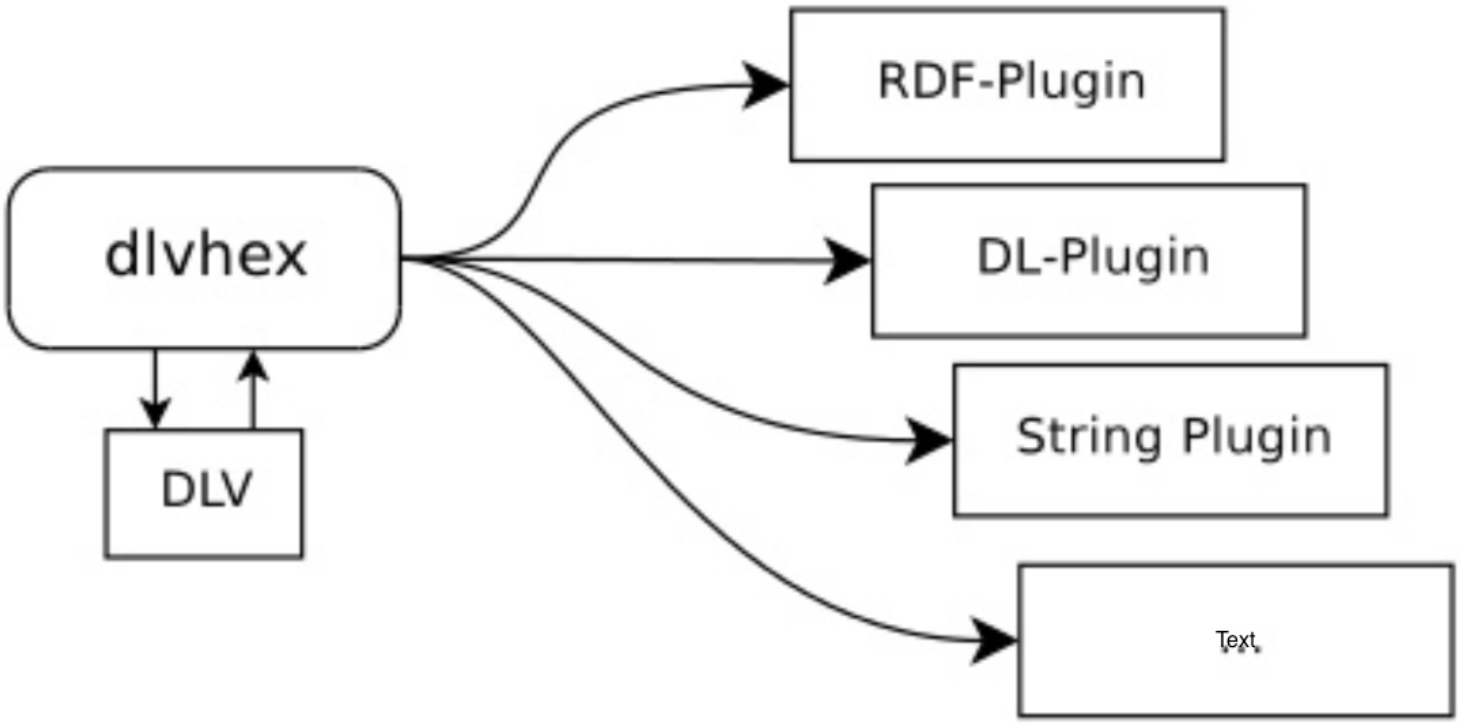
Computation is carried out by compilation to different engines.



c.f. Annual workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP) since 2008

HEX Programs

Higher-order logic programs with EXternal atoms (HEX-programs) [EIL+08]



The program can interface with multiple external sources of knowledge via so called external atoms implemented as "plugins".

Example: HEX Programs

$$triple(X, Y, Z) \leftarrow \&rdf[uri1](X, Y, Z)$$
$$triple(X, Y, Z) \leftarrow \&rdf[uri2](X, Y, Z)$$
$$proposition(P) \leftarrow triple(P, rdf{:}type, rdf{:}Statement)$$

$\&rdf$ is an external predicate intended to extract knowledge from a given URI.

Contents

- General introduction to ASP
- Motivation for ASPMT
- Language of ASPMT
  - Multi-valued propositional formulas
  - First-order formulas
  - Implementations: MVSM and ASPMT2SMT
- High level action language based on ASPMT

ASPMT Motivation

Variables and Grounding

In almost all extensions of ASP language, variables are understood in terms of grounding.

$$p(a)$$
$$q(b)$$
$$r(X) \leftarrow p(X), not\ q(X)$$

is shorthand for the formula

$$p(a)$$
$$q(b)$$
$$r(a) \leftarrow p(a), not\ q(a)$$
$$r(b) \leftarrow p(b), not\ q(b).$$

Grounding is required for applying fixpoint definition.

Grounding approach is widely used: PDDL, inductive logic programming, probabilistic reasoning, etc.

ASP solvers implement intelligent grounding (utilizing minimality of stable models), which produces much less ground instances than the naive way.

Two Sides of Grounding

- (+) Allows for efficient propositional reasoning.
  - Can utilize effective SAT solving methods (CDCL, DPLL).
- (-) Limited to Herbrand models only
- (-) Grounding in the presence of Herbrand functions may not terminate. e.g., $\{a, f(a), f(f(a)), f(f(f(a))), \dots\}$
- (-) "Grounding bottleneck problem": cannot effectively handle a large integer domain, and cannot handle real numbers.

Language Extension: Constraint Answer Set Programs (CASP)

Grounding is often the bottleneck. Solving is not applied until grounding is finished.

To alleviate the grounding bottleneck, integration of ASP with CSP/SMT solvers has been considered.

- Clingcon [GOS09]: Clasp + CSP solver Gecode

$$1 \leq amt(T) \leq 3 \leftarrow pour(T)$$
$$amt(T) = 0 \leftarrow not\ pour(T)$$
$$vol(T+1) = vol(T) + amt(T)$$

- EZCSP [Bal11]: Gringo + constraint solver SICStus Prolog or BProlog
- Dingo [JLN11]: Gringo + SMT solver Barcelogic

Fundamental Limitation Due to Lack of General Functions

(Basic) ASP lacks general functions.

- Functional fluents in ASP are represented by predicates:

$$WaterLevel(t{+}1, tank, l) \leftarrow WaterLevel(t, tank, l),\ not\ \sim WaterLevel(t{+}1, tank, l).$$

  Grounding generates a large number of instances as the domain gets large.

- Using functions (e.g., $WaterLevel(t, tank) = l$) instead does not work because
  - Answer sets are Herbrand models:
    $WaterLevel(t{+}1, tank) = WaterLevel(t, tank)$ is always false.
  - Nonmonotonicity of ASP has to do with minimizing the predicates but has nothing to do with functions.

Fundamental Limitation Due to Lack of General Functions

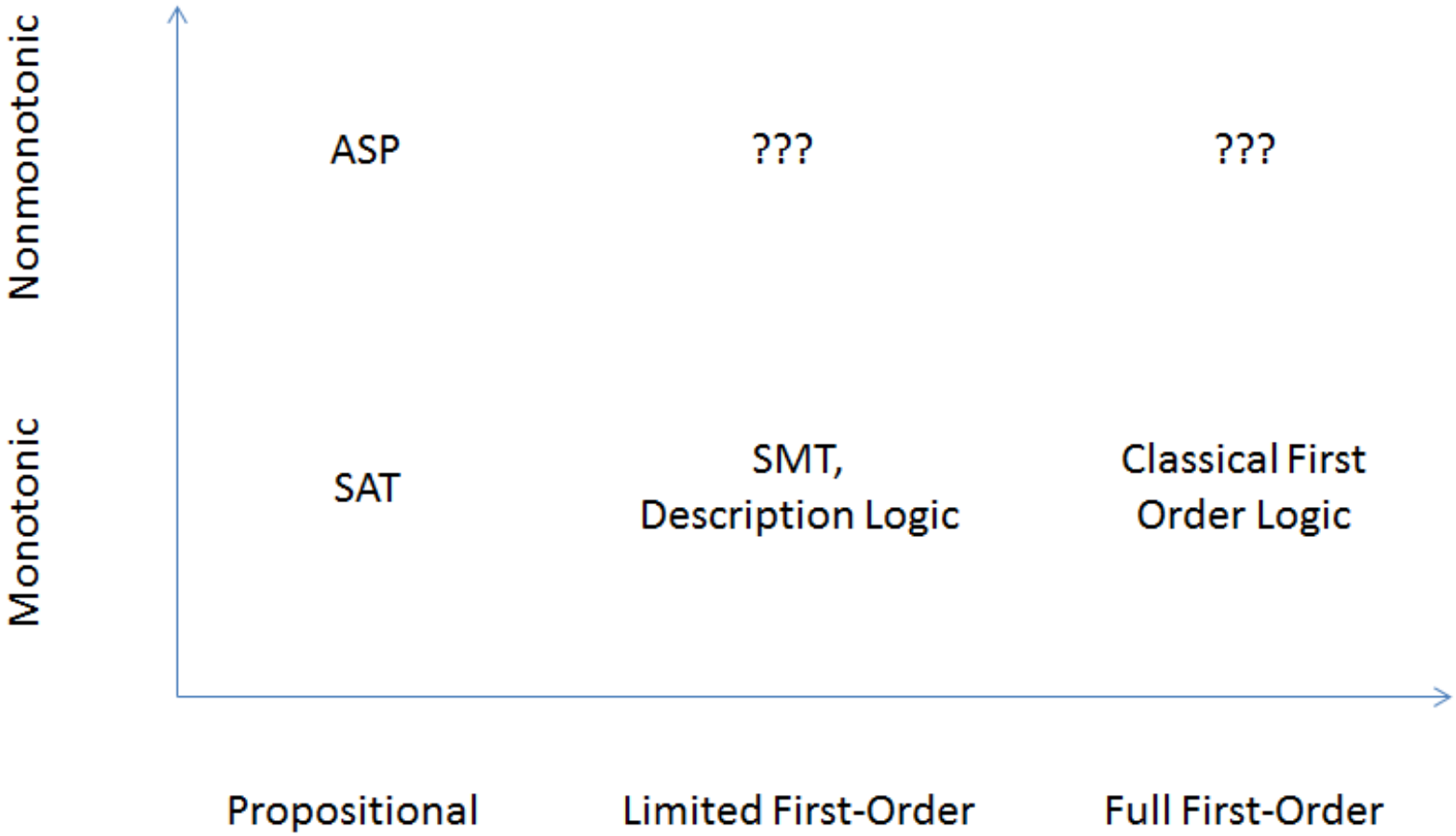
- Even the constraint answer set sovers don't help. In CLINGCON this rule does not affect stable models.

$$\mathit{WaterLevel}(t{+}1, \mathit{tank}) =^{\$} l \leftarrow \mathit{WaterLevel}(t, \mathit{tank}) =^{\$} l,\ \mathit{not}\ \mathit{WaterLevel}(t{+}1, \mathit{tank}) \neq^{\$} l.$$

Lack of general functions in ASP is not only a disadvantage in comparison with other declarative formalisms, but also a hurdle to cross over in integrating ASP with other declarative paradigms where functions are primitive constructs.

Extensions of SAT

We can classify the formalisms based on the two directions the formalisms extend SAT — through extension to first-order reasoning, and through extension to nonmonotonic reasoning



Satisfiability Modulo Theories (SMT)

- SAT is often too restrictive.
- FOL is too general and undecidable.
- Many applications require satisfiability respect to some (decidable) background theory, which fixes the interpretation of certain symbols.
  $\Rightarrow$ Satisfiability modulo background theory

  Some background theories:
  - Difference logic: $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
  - Linear arithmetic over rationals:
    $(k \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.7 \cdot t_0)) \wedge (\neg k \rightarrow (s_1 = s_0))$
  - Non-linear arithmetic over reals:
    $((c = a \cdot b) \wedge (a_1 = a - 1) \wedge (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$
  - Theory of arrays:
    $(b + 2 = c) \wedge f(read(write(a, b, 3), c - 2)) \neq f(c - b + 1)$

SMT-Solvers, SMT-Lib, SMT-Comp

- SMT-Solves: Ario, Barcelogic, CVC, CVC Lite, CVC3, ExtSAT, Harvey, HTP, ICS (SRI), Jat, MathSAT, Sateen, Simplify, STeP, STP, SVC, TSAT, UCLID, Yices (SRI), Zap (Microsoft), Z3 (Microsoft), iSAT
- SMT-Lib: library of benchmarks http://goedel.cs.uiowa.edu/smtlib/
- SMT-Comp: annual SMT-Solver competition

From Wikipedia Article on SMT

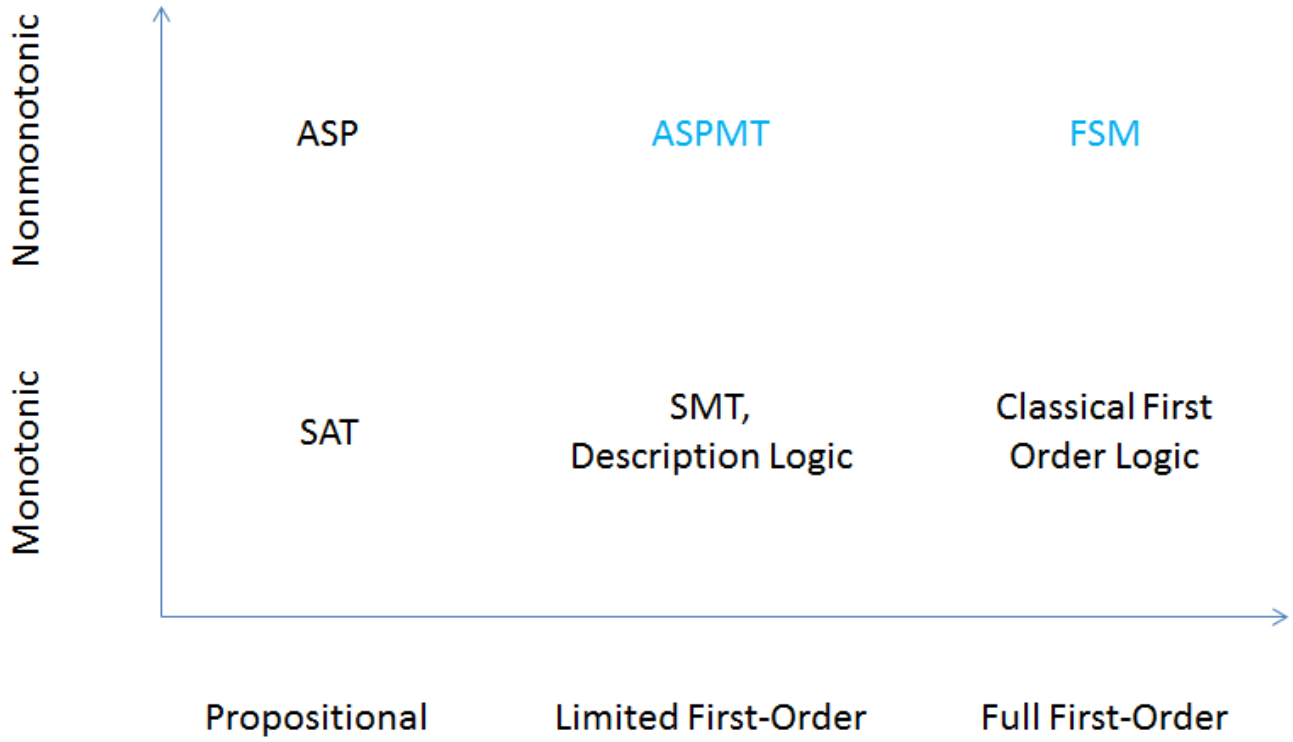
By comparison, answer set programming is also based on predicates *(more precisely, on atomic sentences created from atomic formula). Unlike SMT, answer-set programs do not have quantifiers, and cannot easily express constraints such as linear arithmetic or difference logic–ASP is at best suitable for boolean problems that reduce to the free theory of uninterpreted functions.*

ASP vs. SMT

- ASP is a successful nonmonotonic declarative programming paradigm, but is limited in handling first-order reasoning involving functions due to its propositional setting.
- SMT is a successful approach to solving some specialized first-order reasoning, but is limited in handling expressive nonmonotonic reasoning.

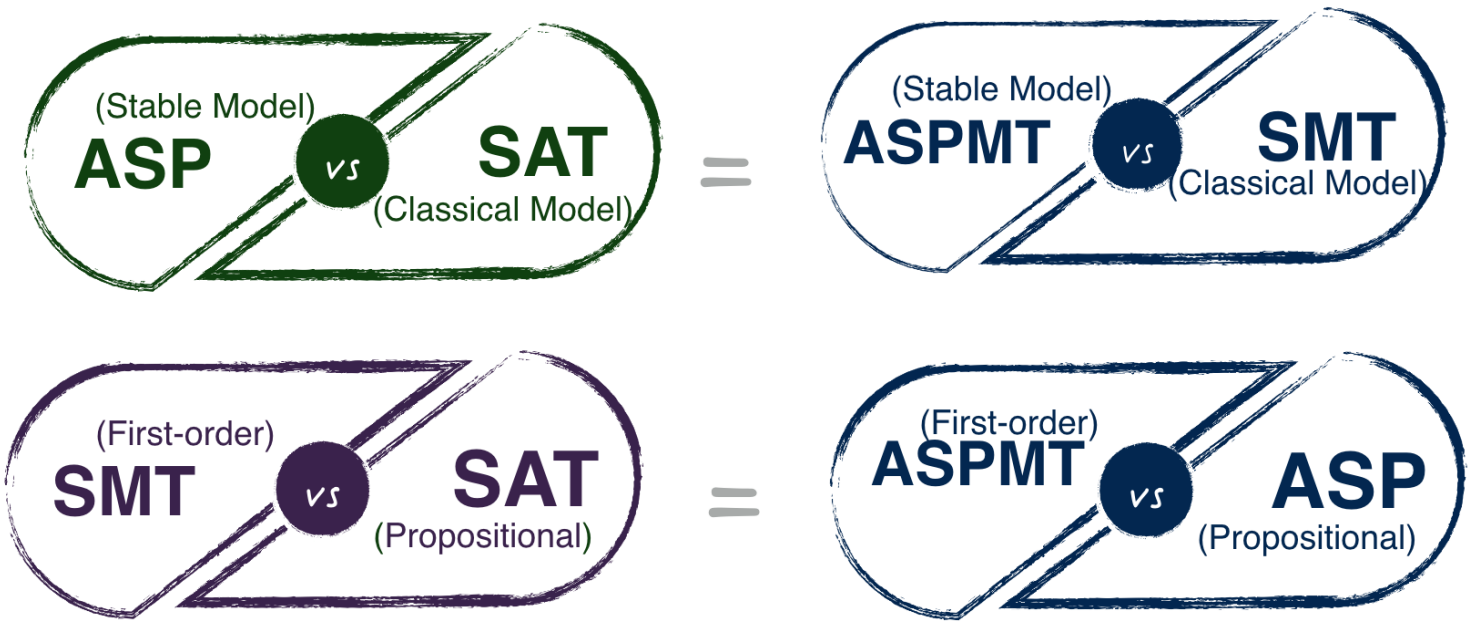
Answer Set Programming Modulo Theories (ASPMT) [Bartholomew and Lee, IJCAI 2013]

ASPMT tightly integrates ASP and SMT.



ASPMT is defined as formulas under the functional stable model semantics (FSM) with the fixed interpretation for the background signature.

ASPMT vs. SMT


(Stable Model)
ASP
vs
SAT
(Classical Model)
=
(Stable Model)
ASPMT
vs
SMT
(Classical Model)
(First-order)
SMT
vs
SAT
(Propositional)
=
(First-order)
ASPMT
vs
ASP
(Propositional)

Related Approaches

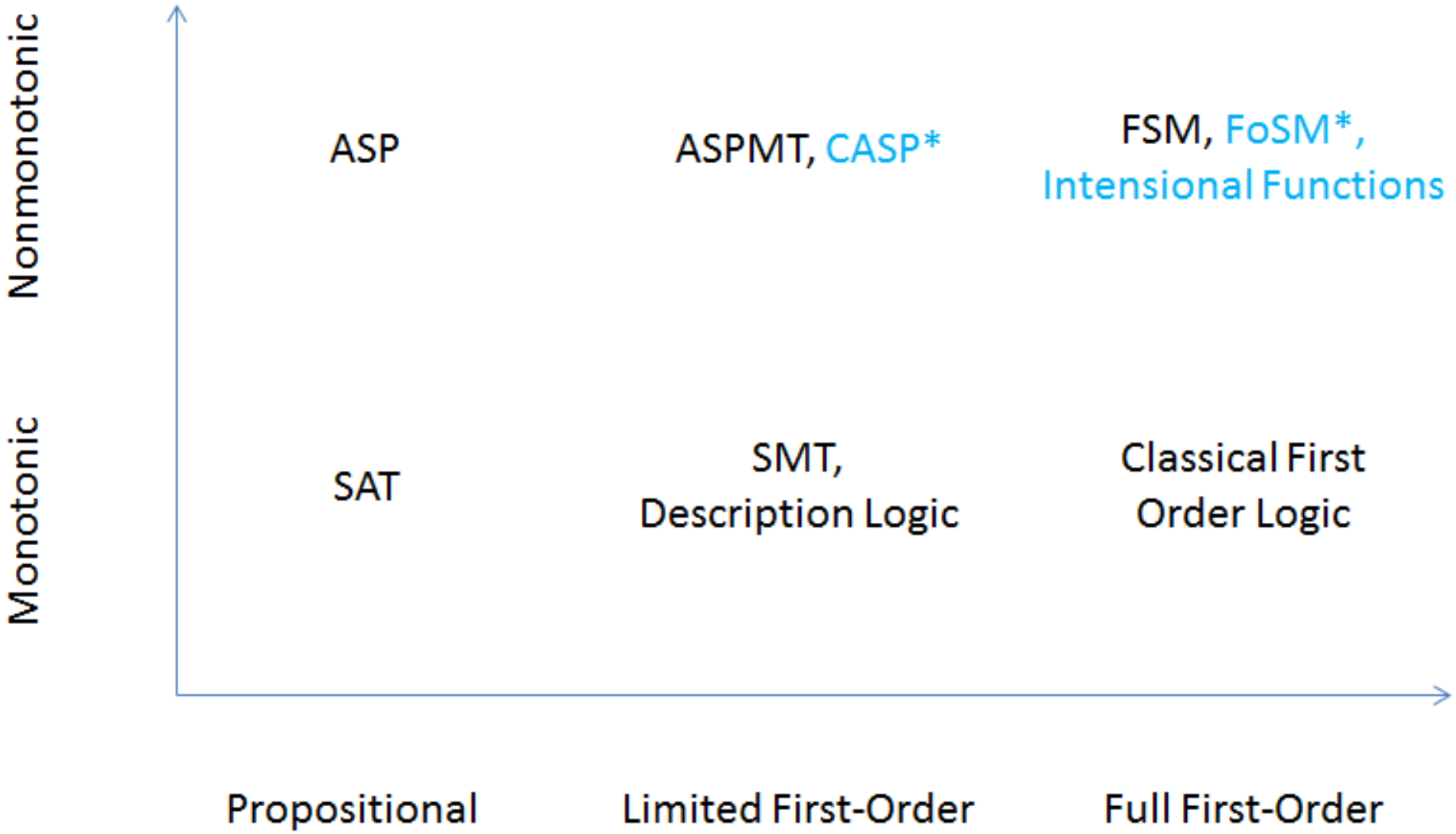
- First-Order Stable Model Semantics (FOSM)
  - In [FLL11], the stable model semantics was extended to the first-order level.
  - Allows non-Herbrand functions.
  - Many useful theoretical and practical results established.
  - Does not allow nonmonotonic reasoning on functions.

Related Approaches

- Intensional Function Proposals [Cabalar, 2011; Lifschitz, 2012; Balduccini, 2012]:
  - Roughly build upon FOSM
  - Perform nonmonotonic reasoning and have expressive functions.
  - Formalism in [Lifschitz, 2012] exhibits some unintuitive behavior.
  - Formalisms in [Cabalar, 2011; Balduccini, 2012] are defined in a complex notation of partial satisfaction.

Related Approaches

These fit into the graph from before as follows.



Related Approaches Summary

The strengths and weaknesses of approaches are summarized in the following table:

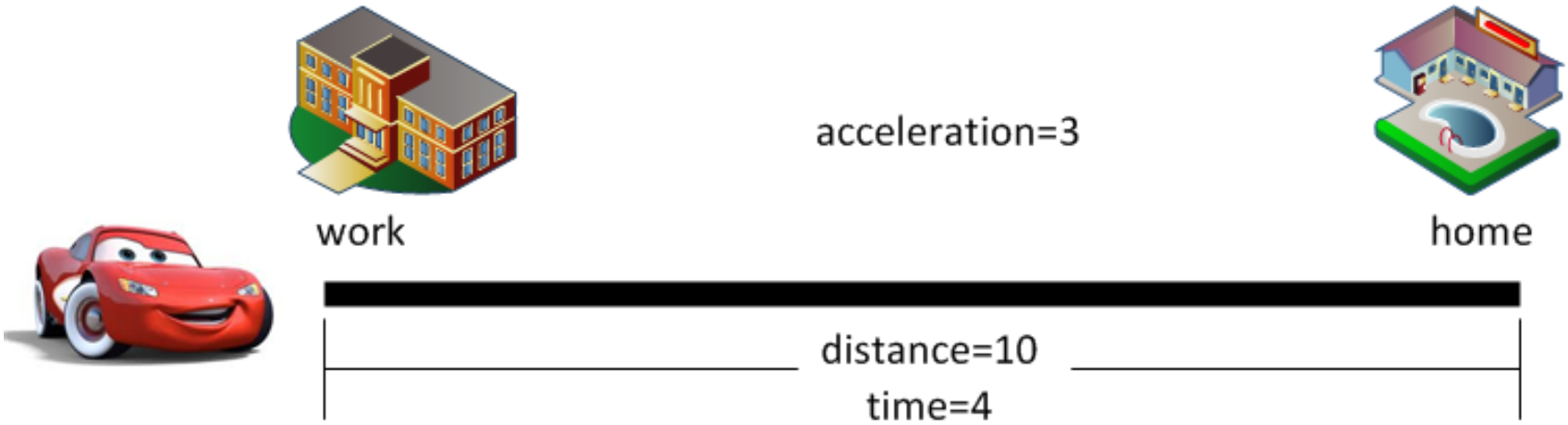
	ASP	SMT	CASP	ASPMT
Non-Herbrand Functions	X	√	√	√
Non-Monotonic Reasoning	√	X	△	√
Alleviates Grounding Bottleneck	X	√	√	√

Examples of ASPMT Domains

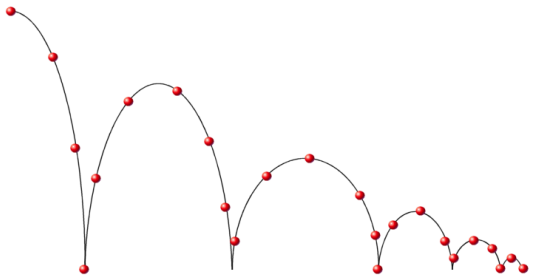
Planning with Continuous Time

[Vladimir Lifschitz, TAG Discussion]

If the accelerator of a car is activated, the car will speed up with constant acceleration A until the accelerator is released or the car reaches its maximum speed MS, whichever comes first. If the brake is activated, the car will slow down with acceleration $-\mathrm{A}$ until the brake is released or the car stops, whichever comes first. Otherwise, the speed of the car remains constant. The problem asks to find a plan satisfying the following condition: at time 0, the car is at rest at one end of the road; at time K, it should be at rest at the other end.



Reasoning about Natural Actions



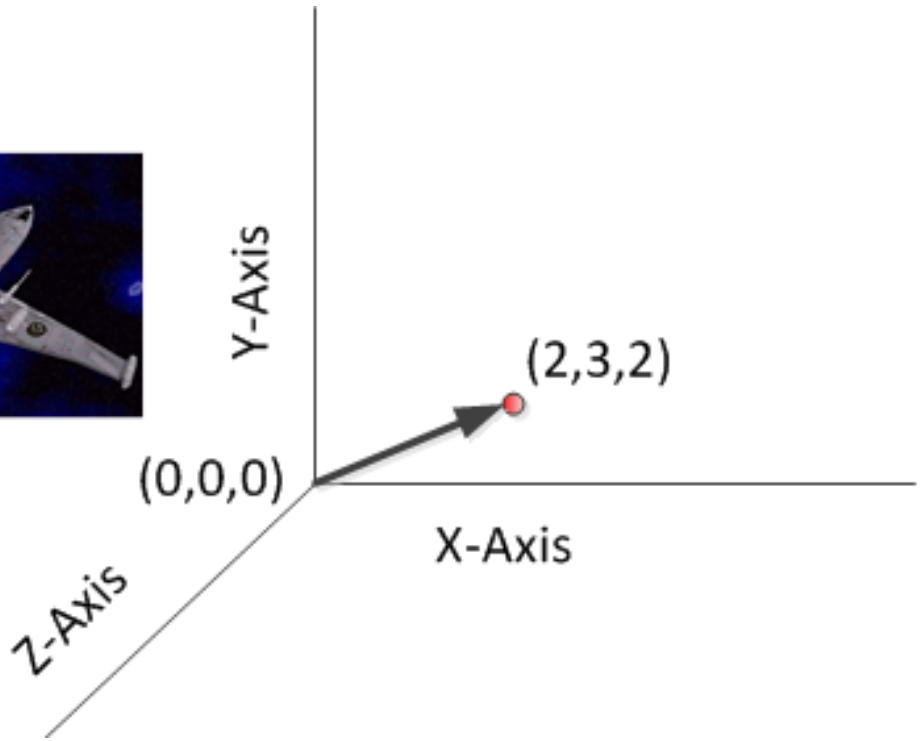
Dropping the ball causes the height of the ball to change continuously with time as defined by Newton's laws of motion.

As the ball accelerates towards the ground it gains velocity. If the ball hits the ground with speed s, it bounces up with speed $r \times s$ where $r = .95$ is the rebound coefficient.

The bouncing ball reaches a certain height and falls back towards the ground due to gravity.

Reasoning about Cumulative Effects on Continuous Changes

A spacecraft has two jets and the force that can be applied by each jet along each axis is at most $4k$. The initial position of the rocket is (0,0,0) and its initial velocity is (0,1,1). How can it get to $(0,3k,2k)$ within 2 seconds?

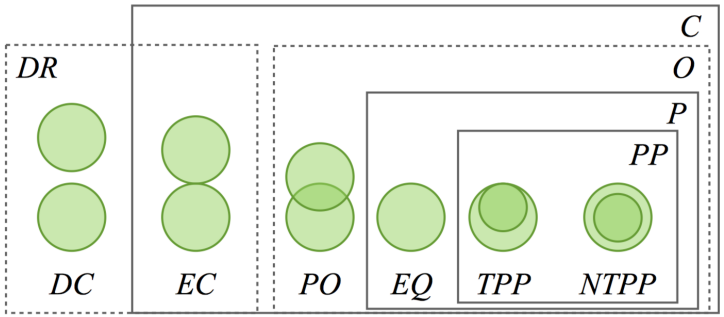


Reasoning about Processes

Consider the kitchen sink with two taps. When a tap is turned on, it starts to fill up. The level of water increases continuously until either both taps are turned off or an outlet is reached.

Nonmonotonic Qualitative Spatial Reasoning

RCC relations between two circular regions:



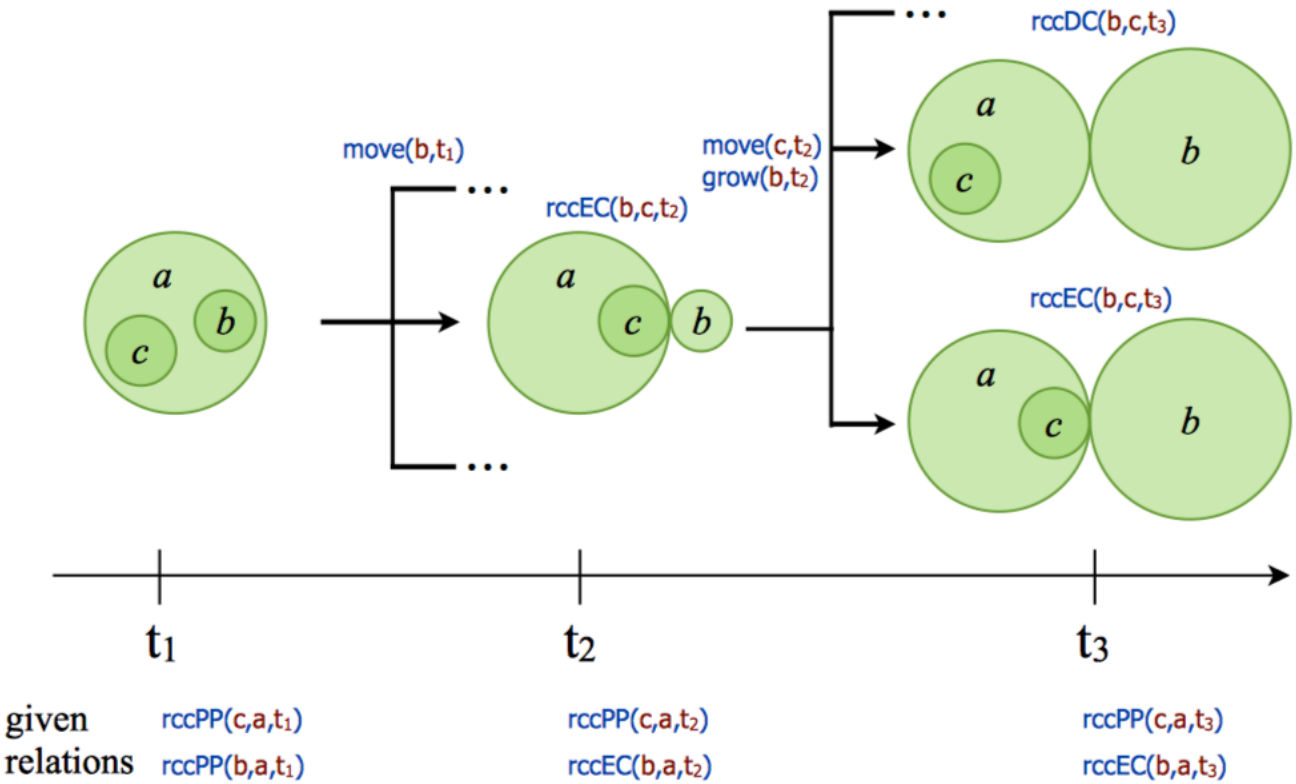
Polynomial encoding of RCC relation:

RCC Relation	**Polynomial Encoding**
contact (C)	$\Delta(c_1, c_2) \leq (r_1 + r_2)^2$
discrete from (DR)	$\Delta(c_1, c_2) \geq (r_1 + r_2)^2$
disconnects (DC)	$\Delta(c_1, c_2) > (r_1 + r_2)^2$
externally connects (EC)	$\Delta(c_1, c_2) = (r_1 + r_2)^2$
overlaps (O)	$\Delta(c_1, c_2) < (r_1 + r_2)^2$
partially overlaps (PO)	$(r_1 - r_2)^2 < \Delta(c_1, c_2) < (r_1 + r_2)^2$
part of (P)	$\Delta(c_1, c_2) \leq (r_1 - r_2)^2 \wedge (r_1 \leq r_2)$
proper part of (PP)	$\Delta(c_1, c_2) \leq (r_1 - r_2)^2 \wedge (r_1 < r_2)$
tangential proper part (TPP)	$\Delta(c_1, c_2) = (r_1 - r_2)^2 \wedge (r_1 < r_2)$
nontangential proper part (NTPP)	$\Delta(c_1, c_2) < (r_1 - r_2)^2 \wedge (r_1 < r_2)$
equal (EQ)	$x_1 = x_2 \wedge y_1 = y_2 \wedge r_1 = r_2$

Nonmonotonic Qualitative Spatial Reasoning

Interval Algebra (Allen, 1983), RCC-5, Cardinal Direction Calculus (Frank, 1991) can be defined in ASPMT [WBS15].

Nonmonotonically inferring actions and qualitative spatial relations



Contents

- General introduction to ASP
- Motivation for ASPMT
- Language of ASPMT
  - Multi-valued propositional formulas
  - First-order formulas
  - Implementations: MVSM and ASPMT2SMT
- High level action language based on ASPMT

Language of ASPMT

Multi-Valued Propositional Formulas

Language of ASPMT

It is apparent that one of the main obstacles encountered in the current work of ASP is due to an insufficient level of generality regarding functions. Solving this problem requires a transformative idea on the concept of a stable model.

We define stable model semantics for

- Multi-valued propositional formulas
- First-order formulas

The former is a special case of the latter. It is simpler to understand.

Syntax: Multi-Valued Propositional Formulas

- *signature* σ : a set of symbols called *constants*

  $\sigma = \{Has, Buy\}$

- $Dom(c)$: a nonempty finite set assigned to each constant c

  $Dom(Has) = \{0, 1, \ldots, 100\}$
  $Dom(Buy) = \{\text{FALSE}, \text{TRUE}\}$

- *atom* : $c = v$ where $c \in \sigma$ and $v \in Dom(c)$

  $Has = 0,\ Has = 1,\ \ldots,\ Has = 100$
  $Buy = \text{TRUE},\ Buy = \text{FALSE}$

- *formula* : propositional combination of atoms

  $Buy = \text{TRUE} \rightarrow Has = 2$

Models of Multi-Valued Propositional Formulas

- An interpretation of σ is a function that maps each $c \in \sigma$ to a value in $Dom(c)$.
- An interpretation I satisfies $c = v$ (symbolically, $I \models c = v$) if $I(c) = v$.
  The satisfaction relation is extended to arbitrary formulas according to the usual truth tables for the propositional connectives.
- I is a model of F if it satisfies F.

  Some tautologies:

  - $c = 1 \vee \neg(c = 1)$
  - $c = 1 \rightarrow (c = 1 \vee d = 2)$
  - $(c = 1 \wedge d = 2) \rightarrow c = 1$

Stable Models of Multi-Valued Propositional Formulas

The stable models of F are defined as the models of F that satisfy the "stability" condition, which is formally defined using the notion of a reduct.

The reduct of F relative to I, denoted by $F^{\underline{I}}$, is the formula obtained from F by replacing each (maximal) subformula that is not satisfied by I with $\bot$.

Example

$\sigma = \{c\}$, $Dom(c) = \{1, 2, 3\}$.
F is $c{=}1 \vee \neg(c{=}1)$.
I_k is an interpretation that maps c to k $(k = 1, 2, 3)$.

- $F^{\underline{I_1}} = (c{=}1 \vee \neg(c{=}1))^{\underline{I_1}} = (c{=}1 \vee \bot) \Leftrightarrow c{=}1$
- $F^{\underline{I_2}} = (c{=}1 \vee \neg(c{=}1))^{\underline{I_2}} = (\bot \vee \neg\bot) \Leftrightarrow \top$

Stable Models of Multi-Valued Propositional Formulas

I is a stable model of F if I is the unique model of the reduct $F^{\underline{I}}$.

Equivalently, I is a stable model of F if

- I satisfies F and
- no other interpretation J satisfies the reduct $F^{\underline{I}}$
  (J disputes the "stability" of I).

In other words, a model of F is stable if it has no witness to dispute the stability for F.

(Closely related to Pearl's causal models and McCain-Turner causal logic)

Example

Example

$\sigma = \{c\}$, $Dom(c) = \{1, 2, 3\}$.
F is $c{=}1 \vee \neg(c{=}1)$.
I_k is an interpretation that maps c to k $(k = 1, 2, 3)$.

- I_1 is a stable model of F.
  $F^{\underline{I_1}} = (c{=}1 \vee \neg(c{=}1))^{\underline{I_1}} = (c{=}1 \vee \bot) \Leftrightarrow c{=}1$
- I_2 is not a stable model of F.
  $F^{\underline{I_2}} = (c{=}1 \vee \neg(c{=}1))^{\underline{I_2}} = (\bot \vee \neg\bot) \Leftrightarrow \top$

Default Formulas

$\{F\}$ stands for $F \vee \neg F$.

$$\{F\}^{\underline{I}} = \begin{cases} F & \text{if } I \models F \\ \top & \text{otherwise.} \end{cases}$$

This construct is useful for expressing defaults.

- $\{c=1\}$ represents that by default c has the value 1.
- $\{c=1\} \wedge c=2$: default is overridden.
  - $(\{c=1\} \wedge c=2)^{\underline{I_1}}$ is equivalent to $c=1 \wedge \bot$, so I_1 is not a stable model of the formula.
  - $(\{c=1\} \wedge c=2)^{\underline{I_2}}$ is equivalent to $\top \wedge c=2$, so I_2 is a stable model of the formula.

This example illustrates nonmonotonicity of the stable model semantics.

Default Formulas and Commonsense Law of Inertia

Default formulas provides a simple and elegant solution to the frame problem.

$$(Loc_0 = L \ \rightarrow \ \{Loc_1 = L\}) \wedge$$

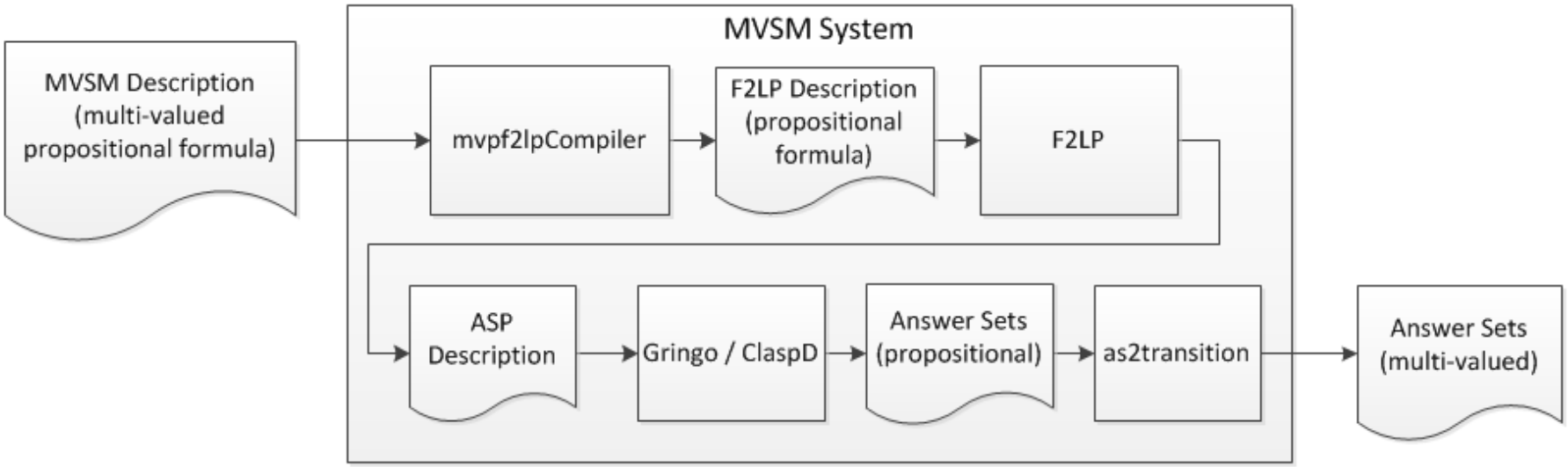
$$(Move(L') \ \rightarrow \ Loc_1 = L')$$

If the location at time 0 is L, by default, the location at time 1 is L.

The default is overridden in the presence of Move.

System MVSM

The language can be turned into the language of ASP solvers [BL12].



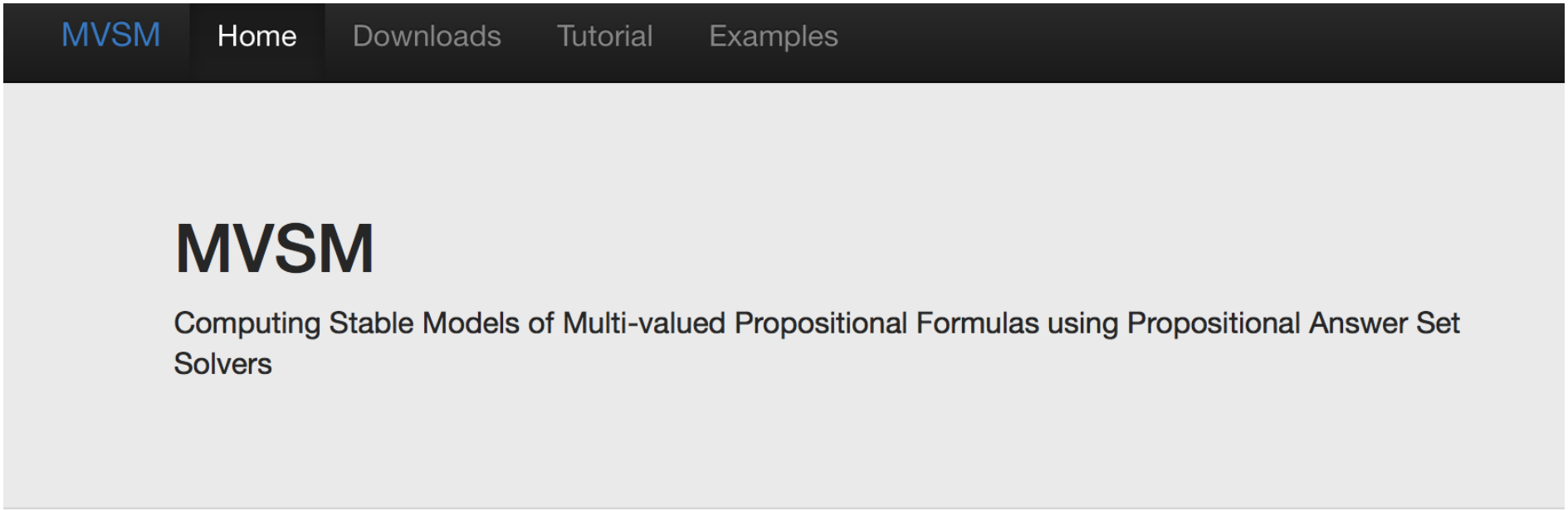
Blocks World in MVSM (I)

```
:- sorts
  step; astep;
  location >> block.

:- objects
  0..maxstep              :: step;
  0..maxstep-1            :: astep;
  1..6                    :: block;
  table                   :: location.

:- variables
  ST                      :: step;
  T                       :: astep;
  Bool                    :: boolean;
  B,B1                    :: block;
  L                       :: location.

:- constants
  loc(block,step)                 :: location;
  move(block,location,astep)      :: boolean.
```

Blocks World in MVSM (II)

```
% two blocks can't be on the same block at the same time
<- loc(B1,ST)=B & loc(B2,ST)=B & B1!=B2.

% effect of moving a block
loc(B,T+1)=L <- move(B,L,T).

% a block can be moved only when it is clear
<- move(B,L,T) & loc(B1,T)=B.

% a block can't be moved onto a block that is being moved also
<- move(B,B1,T) & move(B1,L,T).

% initial location is exogenous
{loc(B,0)=L}.

% actions are exogenous
{move(B,L,T)=Bool}.

% fluents are inertial
{loc(B,T+1)=L} <- loc(B,T)=L.
```

http://reasoning.eas.asu.edu/mvsm



Demo: MVSM

Stable Models of Formulas with Intensional Functions

Functional Stable Model Semantics (FSM) [Bartholomew and Lee, 2012]

The stable model semantics for multi-valued formulas can be generalized to arbitrary first-order formulas by grounding the latter to the former.

The main difference is that since the universe (domain) may be infinite, grounding a first-order sentence F relative to an interpretation I (denoted $gr_I[F]$) may introduce infinite conjunctions and disjunctions.

Leaking Container Example

Describe a water tank that has a leak but that can be refilled to the maximum amount, say 10, with the action *FillUp*.

$$\{Amount_1 = x\} \leftarrow Amount_0 = x+1$$
$$Amount_1 = 10 \leftarrow FillUp\ .$$

$\{F\}$ is a choice rule standing for $F \vee \neg F$

- $I_1 = \{FillUp = \text{FALSE}, Amount_0 = 6, Amount_1 = 5\}$:
  I_1 is a stable model of F (relative to $Amount_1$) as well as a model.
- $I_2 = \{FillUp = \text{FALSE}, Amount_0 = 6, Amount_1 = 8\}$:
  I_2 is a model of F but not a stable model.
- $I_3 = \{FillUp = \text{TRUE}, Amount_0 = 6, Amount_1 = 10\}$:
  I_3 is a model of F as well as a stable model of F.

Infinitary Ground Formula w.r.t. an Interpretation

Since the universe may be infinite, grounding a first-order sentence F relative to an interpretation I (denoted $gr_I[F]$) may introduce infinite conjunctions and disjunctions.

Leaking Container Example. $gr_I[F]$ is

$$\begin{array}{rcl} \{Amount_1 = 0\} & \leftarrow & Amount_0 = 0+1 \\ \{Amount_1 = 1\} & \leftarrow & Amount_0 = 1+1 \\ & \cdots & \\ Amount_1 = 10 & \leftarrow & FillUp \end{array}$$

Reduct-based Definition of FSM

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

- J and I have the same universe and agree on all constants not in $\mathbf{c}$, and
- J and I do not agree on $\mathbf{c}$.

The reduct F^I of an infinitary ground formula F relative to an interpretation I is the formula obtained from F by replacing every maximal subformula that is not satisfied by I with $\bot$.

I is a stable model of F relative to $\mathbf{c}$ (denoted $I \models \mathrm{SM}[F; \mathbf{c}]$) if

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

Leaking Container Example

$I_1 = \{FillUp = \text{FALSE}, Amount_0 = 6, Amount_1 = 5\} \models \text{SM}[F; Amount_1]$

$$
\begin{array}{lrcl}
gr_{I_1}(F): & Amount_1 = 0 \vee \neg(Amount_1 = 0) & \leftarrow & Amount_0 = 0+1 \\
 & \cdots & & \\
 & Amount_1 = 5 \vee \neg(Amount_1 = 5) & \leftarrow & Amount_0 = 5+1 \\
 & \cdots & & \\
 & Amount_1 = 10 & \leftarrow & FillUp \\
\hline
(gr_{I_1}[F])^{\underline{I_1}}: & \bot \vee \neg\bot & \leftarrow & \bot \\
 & \cdots & & \\
 & Amount_1 = 5 \vee \bot & \leftarrow & Amount_0 = 5+1 \\
 & \cdots & & \\
 & \bot & \leftarrow & \bot
\end{array}
$$

No J such that $J \neq^{Amount_1} I_1$ satisfies the reduct.

Leaking Container Example

$I_2 = \{FillUp = \text{FALSE}, Amount_0 = 6, Amount_1 = 8\} \not\models \text{SM}[F; Amount_1]$

$$
\begin{array}{lrcl}
gr_{I_2}(F): & Amount_1 = 0 \vee \neg(Amount_1 = 0) & \leftarrow & Amount_0 = 0+1 \\
 & \cdots & & \\
 & Amount_1 = 5 \vee \neg(Amount_1 = 5) & \leftarrow & Amount_0 = 5+1 \\
 & \cdots & & \\
 & Amount_1 = 10 & \leftarrow & FillUp \\
\hline
(gr_{I_2}[F])^{\underline{I_2}}: & \bot \vee \neg\bot & \leftarrow & \bot \\
 & \cdots & & \\
 & \bot \vee \neg\bot & \leftarrow & Amount_0 = 5+1 \\
 & \cdots & & \\
 & \bot & \leftarrow & \bot
\end{array}
$$

I_2 satisfies the reduct, but there are also other interpretations J such that $J \neq^{Amount_1} I_2$ that satisfy the reduct.

Leaking Container Example

$I_3 = \{FillUp = \text{TRUE}, Amount_0 = 6, Amount_1 = 10\} \models \text{SM}[F; Amount_1]$

$$
\begin{array}{lrcl}
gr_{I_3}(F): & Amount_1 = 0 \vee \neg(Amount_1 = 0) & \leftarrow & Amount_0 = 0+1 \\
 & \cdots & & \\
 & Amount_1 = 5 \vee \neg(Amount_1 = 5) & \leftarrow & Amount_0 = 5+1 \\
 & \cdots & & \\
 & Amount_1 = 10 & \leftarrow & FillUp \\
\hline
(gr_{I_3}[F])^{\underline{I_3}}: & \bot \vee \neg\bot & \leftarrow & \bot \\
 & \cdots & & \\
 & \bot \vee \neg\bot & \leftarrow & Amount_0 = 5+1 \\
 & \cdots & & \\
 & Amount_1 = 10 & \leftarrow & FillUp
\end{array}
$$

No J such that $J <^{Amount_1} I_3$ satisfies the reduct.

FSM in Terms of SOL

c is a list of predicate and function constants called intensional.

u is a list of predicate and function variables corresponding to **c**.

$\mathrm{SM}[F;\ \mathbf{c}]$ is defined as

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

- For predicate symbols (variables or constants) u and c
  - $u \leq c$ is defined as $\forall \mathbf{x}(u(\mathbf{x}) \rightarrow c(\mathbf{x}))$
  - $u = c$ is defined as $\forall \mathbf{x}(u(\mathbf{x}) \leftrightarrow c(\mathbf{x}))$
- For function symbols u and c,
  - $u = c$ is defined as $\forall \mathbf{x}(u(\mathbf{x}) = c(\mathbf{x}))$
- $\mathbf{u} < \mathbf{c}$ is defined as $(\mathbf{u}^{pred} \leq \mathbf{c}^{pred}) \wedge \neg(\mathbf{u} = \mathbf{c})$

FSM in Terms of SOL

The stable models of a first-order sentence F relative to a list of distinct predicate and function constants $\mathbf{c}$ are the models of the second-order formula

$$\mathrm{SM}[F;\ \mathbf{c}] = F \wedge \neg\exists \mathbf{u}(\mathbf{u} < \mathbf{c} \wedge F^*(\mathbf{u}))$$

where $F^*(\mathbf{u})$ is defined as:

- when F is an atomic formula, F^* is $F(\mathbf{u}) \wedge F$;
- $(G \wedge H)^* = G^* \wedge H^*$; $\quad (G \vee H)^* = G^* \vee H^*$;
  $(G \to H)^* = (G^* \to H^*) \wedge (G \to H)$;
- $(\forall x G)^* = \forall x G^*$; $\quad (\exists x F)^* = \exists x F^*$.

Blocks World in FSM

$$
\begin{array}{rcl}
\bot & \leftarrow & Loc(b_1, t) = b \wedge Loc(b_2, t) = b \wedge (b_1 \neq b_2) \\
Loc(b, t+1) = l & \leftarrow & Move(b, l, t) \\
\bot & \leftarrow & Move(b, l, t) \wedge Loc(b_1, t) = b \\
\bot & \leftarrow & Move(b, b_1, t) \wedge Move(b_1, l, t) \\
\{Loc(b, 0) = l\} & & \\
\{Move(b, l, t)\} & & \\
\{Loc(b, t+1) = l\} & \leftarrow & Loc(b, t) = l \ .
\end{array}
$$

The last rule is a default formula that describes the commonsense law of inertia.

Blocks World : Eliminating Function *Loc*

For the class of **c-plain** formulas, intensional functions can be eliminated in favor of intensional predicates.

$$
\begin{array}{rcl}
\bot & \leftarrow & Loc(b_1, b, t) \wedge Loc(b_2, b, t) \wedge \neg(b_1 = b_2) \\
Loc(b, l, t+1) & \leftarrow & Move(b, l, t) \\
\bot & \leftarrow & Move(b, l, t) \wedge Loc(b_1, b, t) \\
\bot & \leftarrow & Move(b, b_1, t) \wedge Move(b_1, l, t) \\
\{Loc(b, l, 0)\} & & \\
\{Move(b, l, t)\} & & \\
\{Loc(b, l, t+1)\} & \leftarrow & Loc(b, l, t)
\end{array}
$$

$$
\begin{array}{rcl}
\bot & \leftarrow & Loc(b, l, t) \wedge Loc(b, l_1, t) \wedge \neg(l = l_1) \\
\bot & \leftarrow & \neg\exists l\ Loc(b, l, t)
\end{array}
$$

Answer Set Programming Modulo Theories (ASPMT)

Defined as a special case of FSM by restricting the attention to interpretations conforming to the background theory.

Let σ^{bg} be the (many-sorted) signature of a background theory $\mathcal{T}$, and let σ be an extension of σ^{bg}.

An interpretation of σ is called a $\mathcal{T}$-interpretation if it agrees with the fixed background interpretation of σ^{bg} according to $\mathcal{T}$.

A $\mathcal{T}$-interpretation I is a $\mathcal{T}$-stable model of F relative to $\mathbf{c}$ if

- $I \models F$ and
- there is no $\mathcal{T}$-interpretation J such that $J \neq^{\mathbf{c}} I$ and $J \models F^{\underline{I}}$.

Completion: Turning ASPMT into SMT

Theorem

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 stable model F iff I is a model of the completion of F.

A formula F is in Clark normal form (relative to $\mathbf{c}$) if it is a conjunction of sentences of the form

$$\forall \mathbf{x} y(G \rightarrow f(\mathbf{x}) = y) \tag{1}$$

one for each function constant f in $\mathbf{c}$, where G is a formula that has no free object variables other than those in $\mathbf{x}$ and y.

We say f depends on g in (1) if g occurs in G but not in the antecedent of any implication in G.

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

Example

Leaking Container Example, Continued.

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

can be rewritten as

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

and completion turns it into

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

The formula can be written without mentioning the variable x:

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

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

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

In the language of SMT solver Z3, this formula can be represented as

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

Comparison with Constraint Answer Set Solving

- CLINGCON programs [GOS09] can be viewed as a special case of ASPMT instances, which allows non-Herbrand functions, but does not allow them to be intensional.
- ASP(LC) programs by [LJN12] can be viewed similarly.
- In fact, they can be viewed even as a special case of the language from [FLL11], which FSM properly generalizes.

Reasoning about Continuous Changes in ASPMT

Planning with Continuous Time

Example: give a formal representation of the domain to generate a plan

acceleration=3

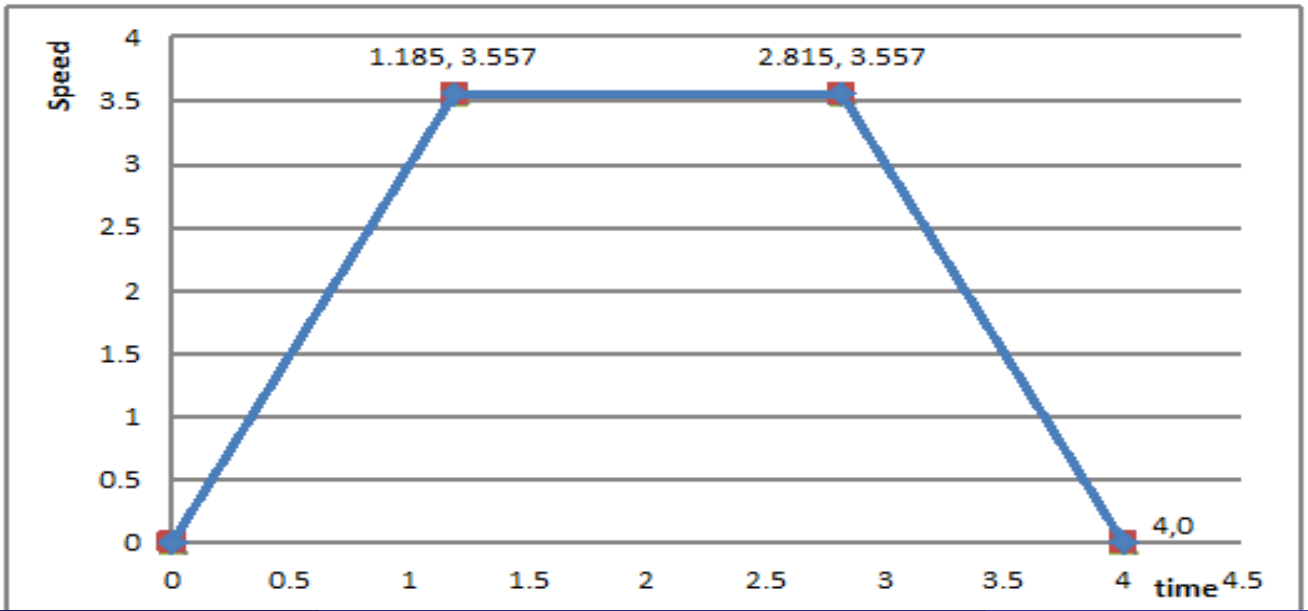
work

home

distance=10

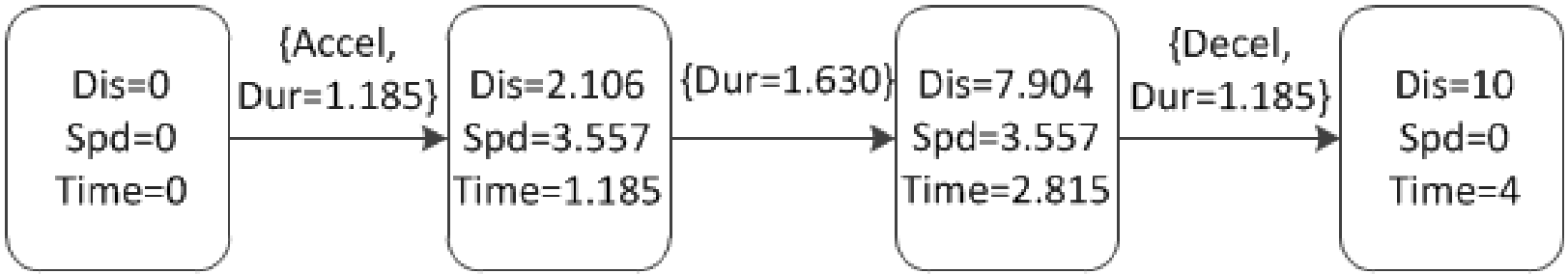
time=4

Solution:



Planning with Continuous Time

Transition System:



We distinguish between steps and real clock times. We assume the Theory of Reals as the background theory, and introduce

- *Time*: fluent with value sort $\mathcal{R}_{\geq 0}$, which denotes real clock time.
- *Dur*: action with value sort $\mathcal{R}_{\geq 0}$, which denotes the time elapsed between the two consecutive states.

Car Example in ASPMT

Intensional constants:

		Domains:
$i:Speed$, $i:Distance$	$(0 \leq i \leq maxstep)$	$\mathcal{R}_{\geq 0}$

Nonintensional constants:

$i:Time$	$(0 \leq i \leq maxstep)$	$\mathcal{R}_{\geq 0}$
$i:Accelerate$, $i:Decelerate$	$(0 \leq i < maxstep)$	Boolean
$i:Dur$	$(0 \leq i < maxstep)$	$\mathcal{R}_{\geq 0}$

Axioms:

$$i{+}1:Speed = v + \mathrm{A} \times t \leftarrow i:(Accelerate \wedge Speed = v \wedge Dur = t)$$
$$i{+}1:Speed = v - \mathrm{A} \times t \leftarrow i:(Decelerate \wedge Speed = v \wedge Dur = t)$$
$$\{i{+}1:Speed = v\} \leftarrow i:Speed = v$$
$$i{+}1:Distance = d + 0.5 \times (v + v') \times t \leftarrow i{+}1:Speed = v' \wedge i:(Distance = d \wedge Speed = v \wedge Dur = t)$$
$$i{+}1:Time = t + t' \leftarrow i:(Time = t \wedge Dur = t')$$

Turning ASPMT into SMT

$$\text{ASPMT} \xrightarrow{\textit{completion}} \text{SMT} \xrightarrow{\textit{eliminating variables}} \text{SMT solvers}$$

In ASPMT:

$$i{+}1 : Speed = x \leftarrow (x = v + \mathrm{A} \times t) \wedge \ i : (Accelerate \wedge Speed = v \wedge Dur = t)$$
$$i{+}1 : Speed = x \leftarrow (x = v - \mathrm{A} \times t) \wedge \ i : (Decelerate \wedge Speed = v \wedge Dur = t)$$
$$i{+}1 : Speed = x \leftarrow \neg\neg(i{+}1 : Speed = x) \wedge i : Speed = x$$

In SMT: The completion on $i{+}1 : Speed$ yields:

$$\begin{aligned} i{+}1 : Speed = x \leftrightarrow \ & \big(x = (i : Speed + A \times i : Dur) \ \wedge \ i : Accelerate\big) \\ & \vee \big(x = (i : Speed - A \times i : Dur) \ \wedge \ i : Decelerate\big) \\ & \vee \big(i{+}1 : Speed = x \ \wedge \ i : Speed = x\big) \end{aligned}$$

In the Language of SMT Solvers

$$i{+}1 : Speed = x \leftrightarrow \big(x = (i : Speed + A \times i : Dur) \ \wedge \ i : Accelerate\big) \\ \vee \ \big(x = (i : Speed - A \times i : Dur) \ \wedge \ i : Decelerate\big) \\ \vee \ \big(i{+}1 : Speed = x \ \wedge \ i : Speed = x\big).$$

Variable x can be eliminated:

$$i : Accelerate \rightarrow i{+}1 : Speed = (i : Speed + A \times i : Dur)$$

$$i : Decelerate \rightarrow i{+}1 : Speed = (i : Speed - A \times i : Dur)$$

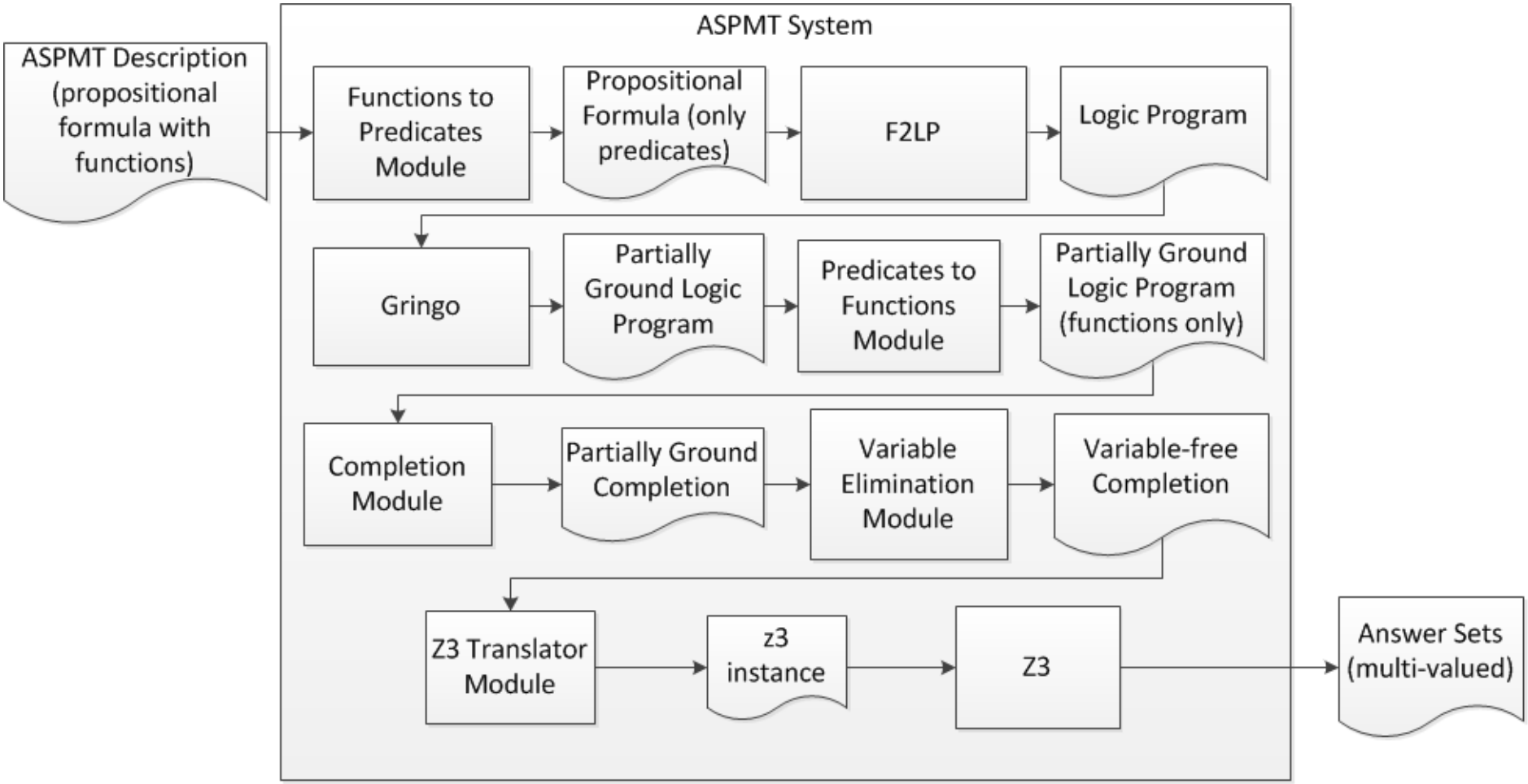
$$(i{+}1 : Speed = (i : Speed + A \times i : Dur) \wedge i : Accelerate) \\ \vee \ (i{+}1 : Speed = (i : Speed - A \times i : Dur) \wedge i : Decelerate) \\ \vee \ (i : Speed = i{+}1 : Speed)$$

System ASPMT2SMT

System ASPMT2SMT

- Tight ASPMT programs can be turned into SMT instances, thereby allowing SMT solvers to compute ASPMT programs.
- We implemented this translation in system ASPMT2SMT, which uses ASP grounder GRINGO and SMT solver Z3. The system can effectively handle real number computation, and compute plans with continuous changes.
  1. Turn formulas into logic program rules.
  2. Partially ground the program by replacing ASP variables with ground terms.
  3. Apply functional completion.
  4. Eliminate SMT variables and invokes a SMT solver.

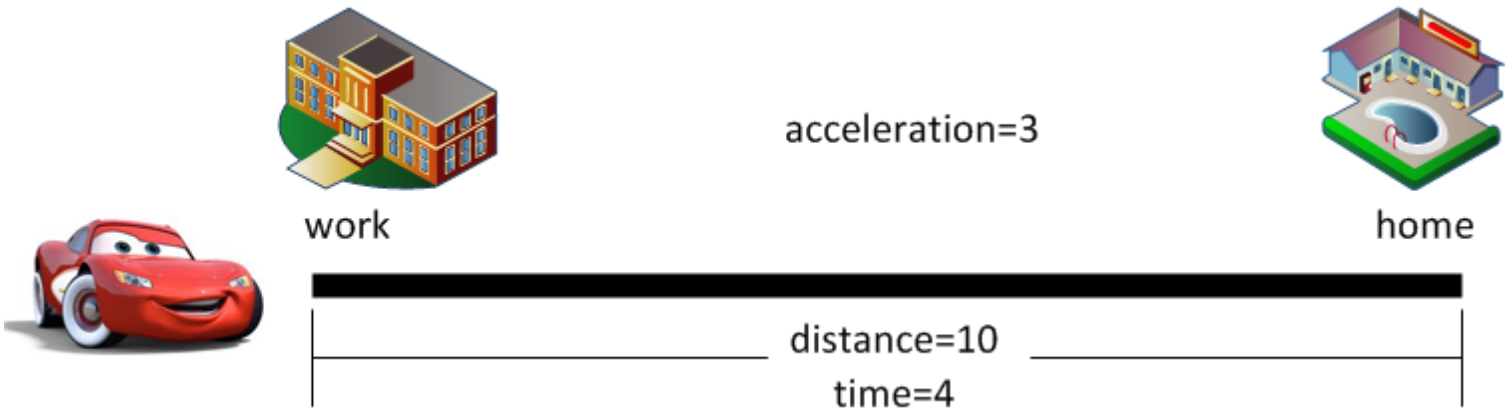
`http://reasoning.eas.asu.edu/aspmt`

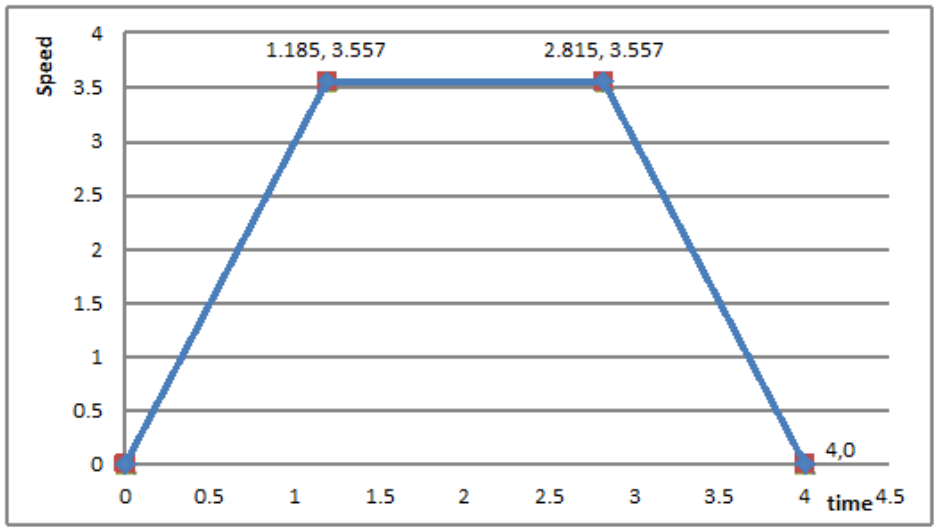


- F2LP: Existing tool that turns formulas into logic programming syntax.
- GRINGO: Existing tool to ground the remaining variables.
- Z3: Existing SMT solver

ASPMT Benchmark: Planning with Continuous Time

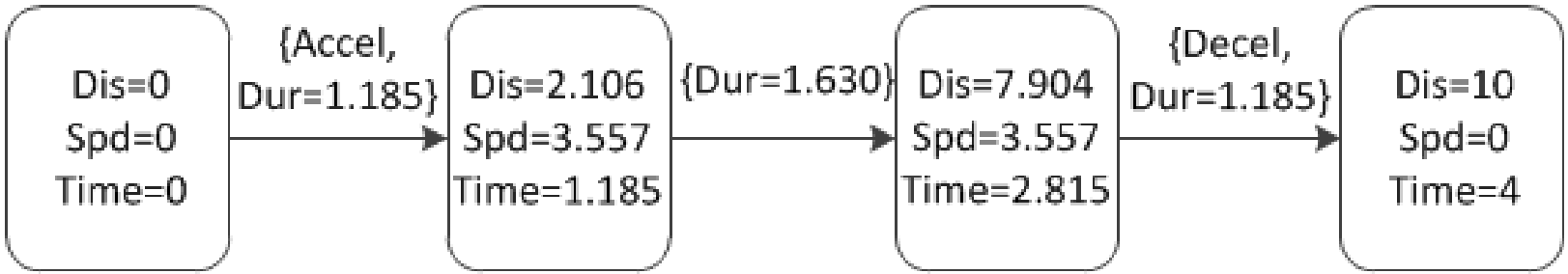
Find a plan satisfying the following condition: at step 0, the car is at rest at one end of the road; at step k, it should be at rest at the other end.





Transition System

The problem is asking to find this path in the transition system:



We distinguish between steps and real clock times. We assume the Theory of Reals as the background theory, and introduce

- *Time*: fluent with value sort $\mathcal{R}_{\geq 0}$, which denotes real clock time.
- *Dur*: action with value sort $\mathcal{R}_{\geq 0}$, which denotes the time elapsed between the two consecutive states.

Car Example in the Language of ASPMT2SMT (I)

```
:- sorts
   step; astep.

:- objects
   0..st             :: step;
   0..st-1           :: astep.

:- constants
   time(step)        :: real[0..t];
   duration(astep)   :: real[0..t];
   accel(astep)      :: boolean;
   decel(astep)      :: boolean;
   speed(step)       :: real[0..ms];
   location(step)    :: real[0..l].

:- variables
   S                 :: astep;
   B                 :: boolean.
```

Car Example in the Language of ASPMT2SMT (II)

```
% Actions and durations are exogenous
{accel(S)=B}.      {decel(S)=B}.      {dur(S)=X}.

% effects of accel and decel
speed(S+1)=Y <- accel(S)=true & speed(S)=X & dur(S)=D & Y = X+ar*D.
speed(S+1)=Y <- decel(S)=true & speed(S)=X & dur(S)=D & Y = X-ar*D.

% preconditions of accel and decel
<- accel(S)=true & speed(S)=X & dur(S)=D & Y = X+ar*D & Y > ms.
<- decel(S)=true & speed(S)=X & dur(S)=D & Y = X-ar*D & Y < 0.

% inertia of speed
{speed(S+1)=X} <- speed(S)=X.

location(S+1)=Y <- location(S)=X & speed(S)=A &
                   speed(S+1)=C & dur(S)=D & Y = X+(A+C)/2*D.

time(S+1)=Y <- time(S)=X & dur(S)=D & Y=X+D.
```

The Input to Z3 Translated by ASPMT2SMT (Part)

```
(assert (= time_S + 1_ (+ time_S_ duration_S_)))
(assert (= time_0_ 0))
(assert (or (or (and (= accel_S_ true) (= speed_S + 1_ (+ speed_S_ (* 3 dur
(assert (=> (= accel_S_ true) (= speed_S + 1_ (+ speed_S_ (* 3 duration_S_)
(assert (=> (= decel_S_ true) (= speed_S + 1_ (- speed_S_ (* 3 duration_S_)
(assert (= speed_0_ 0))
(assert (= location_S + 1_ (+ location_S_ (* (/ (+ speed_S_ speed_S + 1_) 2
(assert (= location_0_ 0))
(assert (not (!= time_3_ 4)))
(assert (not (!= speed_3_ 0)))
(assert (not (!= location_3_ 10)))
(assert (not (and (= decel_S_ true) (< (- speed_S_ (* 3 duration_S_)) 0))))
(assert (not (and (= accel_S_ true) (> (+ speed_S_ (* 3 duration_S_)) 4))))
(assert (not (and (= accel_S_ true) (= decel_S_ true))))
```

Output

This description can be run by the command

```
$aspmt2smt car -c st=3 -c t=4 -c ms=4 -c ar=3 -c l=10
```

which yields the output

```
  accel(0) = true  accel(1) = false  accel(2) = false
  decel(0) = false  decel(1) = false  decel(2) = true
  dur(0) = 1.1835034190  dur(1) = 1.6329931618  dur(2) = 1.1835034190
  location(0) = 0.0 location(1) = 2.1010205144
        location(2) = 7.8989794855  location(3) = 10.0
  speed(0) = 0.0  speed(1) = 3.5505102572  speed(2) = 3.5505102572
        speed(3) = 0.0
  time(0) = 0.0  time(1) = 1.1835034190 time(2) = 2.8164965809
        time(3) = 4.0
z3 time in milliseconds: 30
Total time in milliseconds: 71
```

Experiment: Car Example

L $= 10k$, A $= 3k$, MS $= 4k$, T $= 4k$, which yields solutions with irrational values and so cannot be solved by system CLINGO.

k	CLINGO v3.0.5 Run Time (Grounding + Solving)	ASPMT2SMT v0.9 Run Time (Preprocessing + solving)
1	n/a	.084s (.054s + .03s)
5	n/a	.085s (.055s + .03s)
10	n/a	.085s (.055s + .03s)
50	n/a	.087s (.047s + .04s)
100	n/a	.088s (.048s + .04s)

Experiment: Car Example

L $= 4k$, A $= k$, MS $= 4k$, T $= 4k$, which yields solutions with integral values and so can be solved by system CLINGO.

k	CLINGO v3.0.5 Run Time (Grounding + Solving)	ASPMT2SMT v0.9 Run Time (Preprocessing + solving)
1	.61s (.6s + .01s)	.060s (.050s + .01s)
2	48.81s (48.73s + .08s)	.07s (.050s + .02s)
3	$>$ 30 minutes	.072s (.052s + .02s)
5	$>$ 30 minutes	.068s (.048s + .02s)
10	$>$ 30 minutes	.068s (.048s + .02s)
50	$>$ 30 minutes	.068s (.048s + .02s)
100	$>$ 30 minutes	.072s (.052s + .02s)

In this example, only the SMT variables have increasing domains but the ASP variable domain remains the same. Consequently, the ASPMT2SMT system scales very well compared to the ASP system which can only complete the two smallest size domains.

Experiment: Space Shuttle

k	CLINGO v3.0.5 Run Time (Grounding + Solving)	ASPMT2SMT v0.9 Run Time (Preprocessing + solving)
1	0s (0s + 0s)	.048s (.038s + .01s)
5	.03s (.02s + .01s)	.047s (.037s + .01s)
10	.14s (.9s + .5s)	.053s (.043s + .01s)
50	7.83s (3.36s + 4.47s)	.050s (.040s + .01s)
100	39.65s (16.14s + 23.51s)	.051s (.041s + .01s)

Experiments: Bouncing Ball

k	CLINGO v3.0.5 Run Time (Grounding + Solving)	ASPMT2SMT v0.9 Run Time (Preprocessing + solving)
1	n/a	.072s (.062s + .01s)
10	n/a	.072s (.062s + .01s)
100	n/a	.071s (.061s + .01s)
1000	n/a	.075s (.065s + .01s)
10000	n/a	.082s (.062s + .02s)

http://reasoning.eas.asu.edu/aspmt

ASPMT2SMT Home Downloads Tutorial Examples

ASPMT2SMT

Computing ASPMT Theories using SMT Solvers

System aspmt2smt is a prototype implementation of multi-valued propositional formulas under the stable model semantics computed by the SMT solver Z3. This reduction is based on the theorem on completion which describes how to capture the non-monotonic semantics of ASPMT in classical logic. The system is a toolchain that includes aspmt-compiler, f2lp, gringo, and z3.

The implementation first compiles the ASPMT theory into a first-order formula without functions. F2lp is used to turn these first-order formulas into normal logic programs. Gringo is then used to partially ground the logic program. The system then converts the logic program back into an ASPMT theory with functions that is now partially ground. Then, the system computes the completion of the partially ground ASPMT theory, eliminates any remaining variables resulting in a variable-free first order formula with function. Finally, Z3 computes the classical models of this first-order formula, which correspond to the stable models of the

Demo: ASPMT2SMT

Contents

- General introduction to ASP
- Motivation for ASPMT
- Language of ASPMT
  - Multi-valued propositional formulas
  - First-order formulas
  - Implementations: MVSM and ASPMT2SMT
- High level action language based on ASPMT

High Level Action Language Based on ASPMT

Action Languages

- Action languages are high level languages that allow us to represent knowledge about actions concisely.
- Action description contains a set of causal laws, such as

$$Move(x, y) \textbf{ causes } Loc(x) = y$$

  which defines a transition system.
- A transition system is a directed graph. Its vertices represent states of world. Its edges represent execution of actions.
- Many action languages are defined as high level notations of nonmonotonic logics.
  - $\mathcal{A}$, $\mathcal{B}$, $\mathcal{BC}$, $\mathcal{AL}$, $\mathcal{K}$, ... : in terms of logic programs under the stable model semantics.
  - $\mathcal{C}$ and $\mathcal{C}+$: in terms of nonmonotonic causal theories.

Action Language $\mathcal{C}+$ [GLL+04]

- $\mathcal{C}+$ is a formal model of parts of natural language for representing and reasoning about transition systems.
- Can represent actions with conditional and indirect effects, nondeterministic actions, and concurrently executed actions.
- Can represent multi-valued fluents, defined fluents, additive fluents, and rigid constants.
- Can represent defeasible causal laws and action attributes.
- Implemented in systems CCALC, CPLUS2ASP, COALA.
- New generations: $\mathcal{BC}$ [LLY13], $\mathcal{BC}+$ [BL14].

Action Language $\mathcal{BC}+$ [BL14]

- Successor of $\mathcal{C}+$ and $\mathcal{BC}$ [Lee, Lifschitz and Yang, IJCAI 2013]. Generalizes both $\mathcal{B}$ and $\mathcal{C}+$.
- The main idea is to define the semantics of $\mathcal{BC}+$ in terms of formulas under the stable model semantics [Ferraris, 2005].
- Modern ASP language constructs, such as choice rules and aggregates, can be viewed as an abbreviation of formulas under the stable model semantics.
- Enhancements in ASP are readily applied in the setting of action languages: online answer set solving, ASPMT, interface with external evaluation.

Syntax of $\mathcal{BC}+$

We consider propositional formulas whose signature consists of atoms of the form $c = v$, where c is called a constant and is associated with a finite set called the domain. Constants are either fluents or actions.

$\{c = v\}$ stands for $(c = v) \vee \neg(c = v)$.

Intuitive reading: "by default, c has the value v."

- Static law: **caused** F **if** G (F, G are fluent formulas)

"The light is usually on while the switch is on":

caused $\{Light = On\}$ **if** $Switch = On$

Alternatively: **default** $Light = On$ **if** $Switch = On$

Syntax of $\mathcal{BC}+$

- Action dynamic law: **caused** F **if** G (F is an action formula)

"The agent may move to arbitrary locations":

caused $\{Move = l\}$ **if** $\top$ (for all $l \in Locations$)

Alternatively: **exogenous** *Move*

Syntax of $\mathcal{BC}+$

- Fluent dynamic law: **caused** F **if** G **after** H (F, G: fluent formulas)

  The effect of *Move*:

  caused $Loc = l$ **if** $\top$ **after** $Move = \text{TRUE}$

  Alternatively: *Move* **causes** $Loc = l$

  "The agent's location is inertial":

  caused $\{Loc = l\}$ **if** $\top$ **after** $Loc = l$ (for all $l \in Locations$)

  Alternatively:

  inertial Loc

Semantics of $\mathcal{BC}+$

- For every action description D in $\mathcal{BC}+$ we define a sequence of formulas $PF_0(D), PF_1(D), \ldots$ so that the stable models of $PF_m(D)$ represent paths of length m in the transition system.
- The signature of $PF_m(D)$ consists of the pairs $i:c$ such that
  - $i \in \{0, \ldots, m\}$ and c is a fluent constant of D, and
  - $i \in \{0, \ldots, m-1\}$ and c is an action constant of D.

D	$PF_m(D)$
caused F **if** G	$i:F \leftarrow i:G$
caused F **if** G **after** H	$i+1:F \leftarrow (i+1:G) \wedge (i:H)$
	$\{0:c=v\}$ for every regular fluent c and every v

Transition System Described by D

- A state is an interpretation s of fluent constants such that $0:s$ is a stable model of $PF_0(D)$.
- A transition is a triple $\langle s, e, s' \rangle$ such that s and s' are interpretations of fluent constants and e is an interpretation of action constants such that $0:s \cup 0:e \cup 1:s'$ is a stable model of $PF_1(D)$.

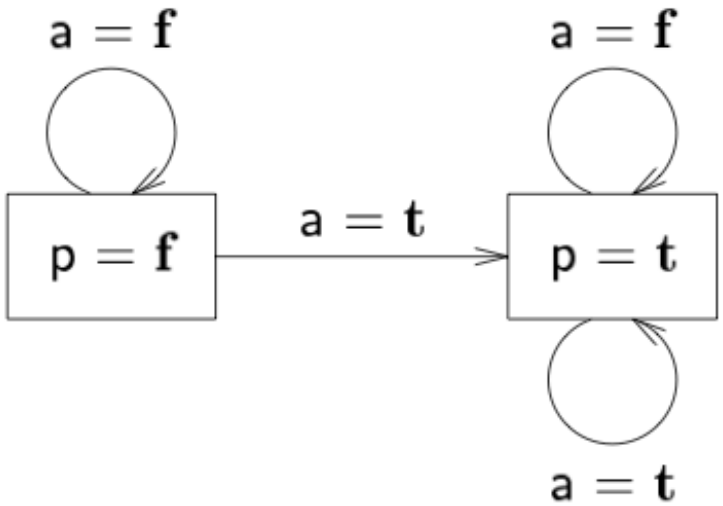
Theorem

The stable models of $PF_m(D)$ are in a 1-1 correspondence with the paths of length m in the transition system D.

Useful Abbreviations

a **causes** F **if** G	$\mapsto$	**caused** F **after** $a \wedge G$
nonexecutable F **if** G	$\mapsto$	**caused** $\bot$ **if** $\top$ **after** $F \wedge G$
default $c = v$	$\mapsto$	**caused** $\{c = v\}$
exogenous c	$\mapsto$	**default** $c = v$ (for all $v \in Dom(c)$)
inertial c	$\mapsto$	**default** $c = v$ **after** $c = v$ (for all $v \in Dom(c)$)
constraint F	$\mapsto$	**caused** $\bot$ **if** $\neg F$

A Simple Transition System in $\mathcal{BC}+$



D	$PF_m(D)$ $\quad (i = 0, \ldots, m-1)$
a **causes** p	$i+1 : p = \text{TRUE} \leftarrow i : a = \text{TRUE}$
exogenous a	$\{i : a = Bool\}$
inertial p	$\{i+1 : p = Bool\} \leftarrow i : p = Bool$
	$\{0 : p = Bool\}$

Blocks World in $\mathcal{BC}+$ (I)

The definition of $\mathit{InTower}(B)$:

$$\begin{array}{l}\textbf{caused } \mathit{InTower}(B) \textbf{ if } \mathit{Loc}(B) = \mathit{Table} \\ \textbf{caused } \mathit{InTower}(B) \textbf{ if } \mathit{Loc}(B) = B_1 \wedge \mathit{InTower}(B_1) \\ \textbf{default } \mathit{InTower}(B) = \text{FALSE}\end{array}$$

Blocks don't float in the air:

$$\textbf{constraint } \mathit{InTower}(B)$$

No two blocks are on the same block:

$$\textbf{constraint } \mathit{Loc}(B_1) = B \wedge \mathit{Loc}(B_2) = B \quad (B_1 \neq B_2)$$

Blocks World in $\mathcal{BC}+$ (II)

The effect of moving a block:

$$\mathit{Move}(B, L) \textbf{ causes } \mathit{Loc}(B) = L.$$

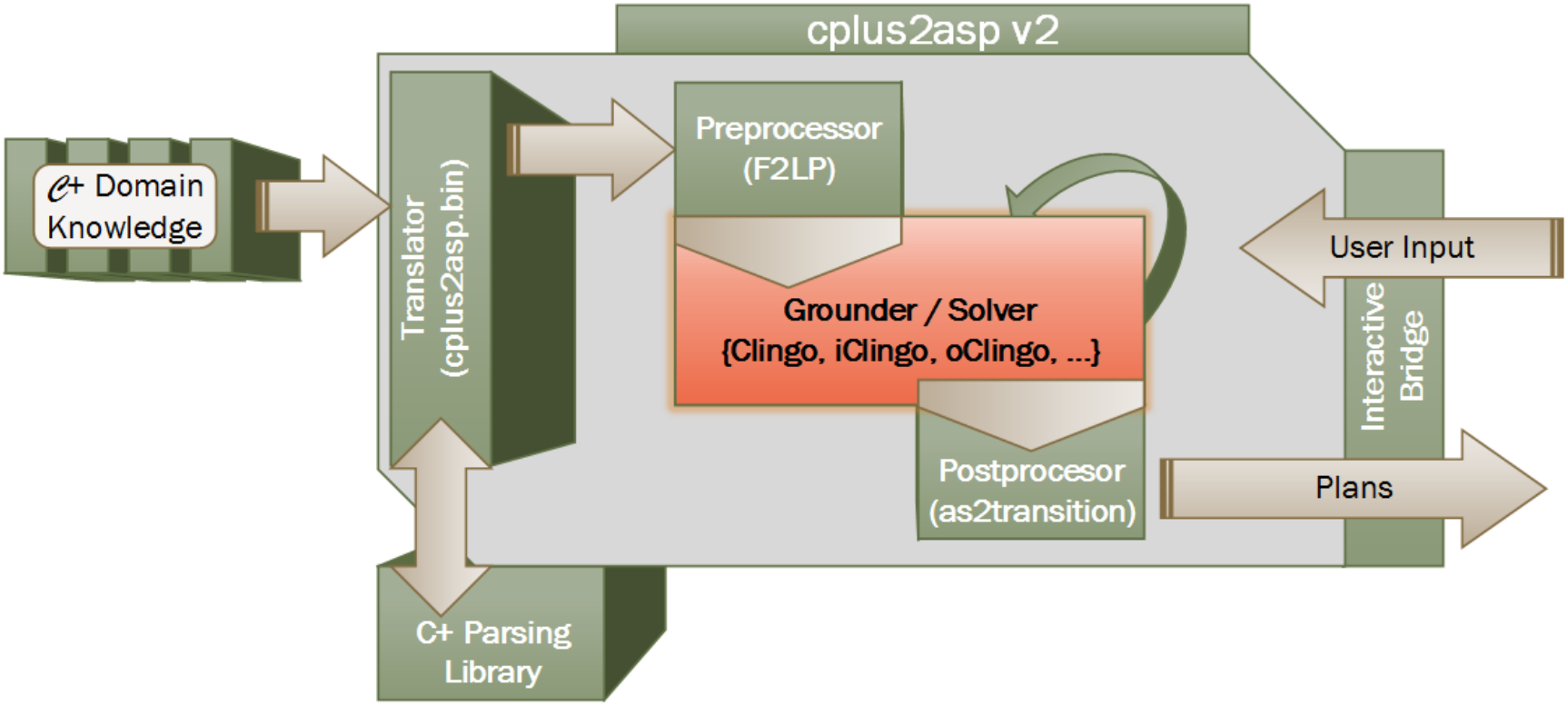
A block cannot be moved unless it is clear:

$$\textbf{nonexecutable } \mathit{Move}(B, L) \textbf{ if } \mathit{Loc}(B_1) = B.$$

The commonsense law of inertia:

$$\textbf{inertial } \mathit{Loc}(B).$$

Implementation



Homepage: `http://reasoning.eas.asu.edu/cplus2asp`

High Level Action Language Based on ASPMT

Representing Continuous Changes in $\mathcal{BC}+$

We distinguish between steps and real clock times. We assume the Theory of Reals as the background theory, and introduce

- *Time*: a simple fluent constant with value sort $\mathcal{R}_{\geq 0}$ (clock time);
- *Dur*: an action constant with value sort $\mathcal{R}_{\geq 0}$, which denotes the time elapsed between the two consecutive states.

We postulate:

$$\begin{array}{l}\textbf{caused } \mathit{Time} = t \textbf{ if } \mathit{Time} = t \\ \textbf{caused } \mathit{Dur} = t \textbf{ if } \mathit{Dur} = t \\ \textbf{caused } \bot \textbf{ if } \neg(\mathit{Time} = t + t') \textbf{ after } \mathit{Time} = t \wedge \mathit{Dur} = t'\end{array}$$

Continuous changes can be described as a function of duration using fluent dynamic laws

$$\textbf{caused } c = f(\mathbf{x}, \mathbf{x}', t) \textbf{ if } \mathbf{c}' = \mathbf{x}' \textbf{ after } (\mathbf{c} = \mathbf{x}) \wedge (\mathit{Dur} = t) \wedge G$$

Planning with Continuous Time

Example: give a formal representation of the domain to generate a plan.

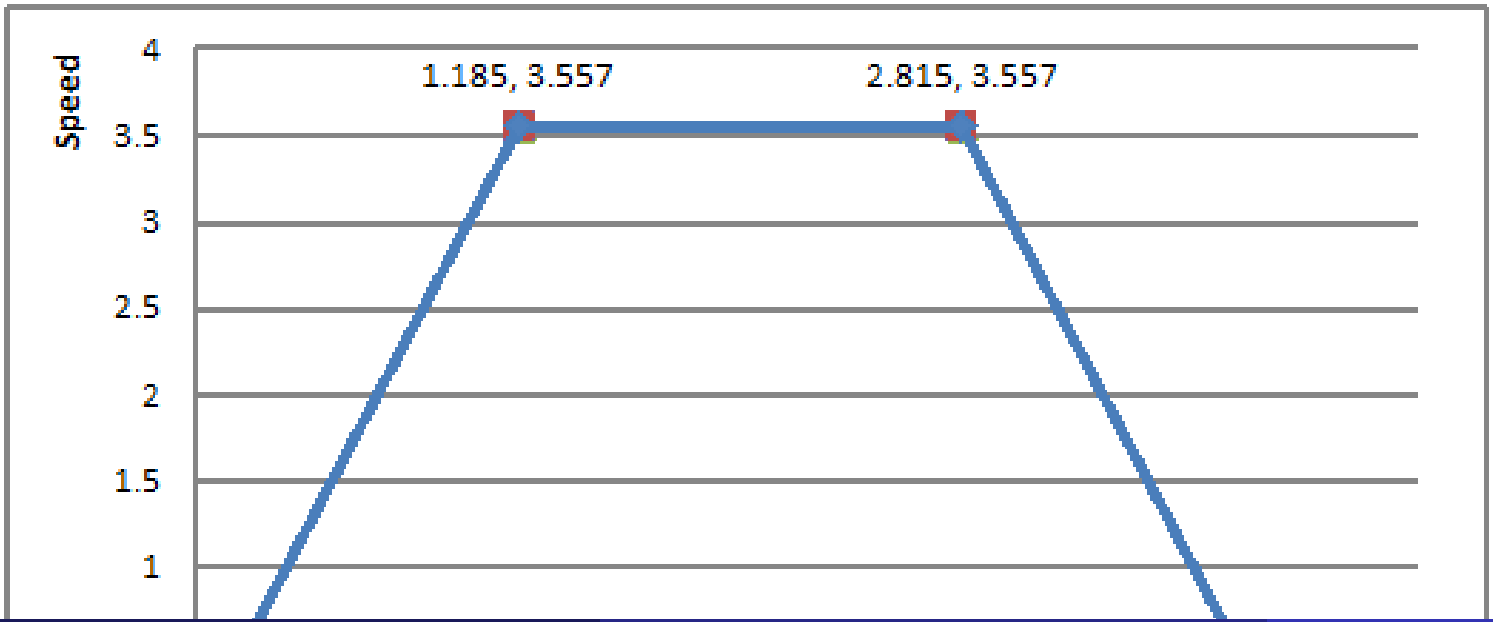
acceleration=3

work

home

distance=10

time=4



Car Example in $\mathcal{BC}+$

Notation: d, v, v', t, t' are variables of sort $\mathcal{R}_{\geq 0}$; A, MS are real numbers.

Simple fluent constants:
- *Speed*, *Distance*, *Time*

Domains:
- $\mathcal{R}_{\geq 0}$

Action constants:
- *Accelerate*, *Decelerate*
- *Dur*

Domains:
- Boolean
- $\mathcal{R}_{\geq 0}$

Causal laws:

caused $\mathit{Speed} = v + \mathrm{A} \times t$ **after** $\mathit{Accelerate} \wedge \mathit{Speed} = v \wedge \mathit{Dur} = t$

caused $\mathit{Speed} = v - \mathrm{A} \times t$ **after** $\mathit{Decelerate} \wedge \mathit{Speed} = v \wedge \mathit{Dur} = t$

caused $\mathit{Distance} = d + 0.5 \times (v + v') \times t$ **if** $\mathit{Speed} = v'$
after $\mathit{Distance} = d \wedge \mathit{Speed} = v \wedge \mathit{Dur} = t$

constraint $\mathit{Time} = t + t'$ **after** $\mathit{Time} = t \wedge \mathit{Dur} = t'$

constraint $\mathit{Speed} \leq \mathrm{MS}$

inertial *Speed*

exogenous *Time*

exogenous c for every action constant c

Turning $\mathcal{BC}+$ into ASPMT and SMT

$$\mathcal{BC}+ \xrightarrow{\textit{semantics}} \text{ASPMT} \xrightarrow{\textit{completion}} \text{SMT} \xrightarrow{\textit{eliminating variables}} \text{SMT solvers}$$

In $\mathcal{BC}+$:

caused $Speed = v + \mathrm{A} \times t$ **after** $Accelerate \wedge Speed = v \wedge Dur = t$
caused $Speed = v - \mathrm{A} \times t$ **after** $Decelerate \wedge Speed = v \wedge Dur = t$
caused $Speed = v$ **if** $Speed = v$ **after** $Speed = v$

In ASPMT:

$i+1 : Speed = x \leftarrow (x = v + \mathrm{A} \times t) \wedge i : (Accelerate \wedge Speed = v \wedge Dur = t)$
$i+1 : Speed = x \leftarrow (x = v - \mathrm{A} \times t) \wedge i : (Decelerate \wedge Speed = v \wedge Dur = t)$
$i+1 : Speed = x \leftarrow \neg\neg(i+1 : Speed = x) \wedge i : Speed = x$

In SMT: The completion on $i+1 : Speed$ yields a formula that is equivalent to

$$\begin{aligned} i+1 : Speed = x \leftrightarrow\ & \big(x = (i : Speed + A \times i : Dur) \ \wedge \ i : Accelerate\big) \\ & \vee \big(x = (i : Speed - A \times i : Dur) \ \wedge \ i : Decelerate\big) \\ & \vee \big(i+1 : Speed = x \ \wedge \ i : Speed = x\big). \end{aligned}$$

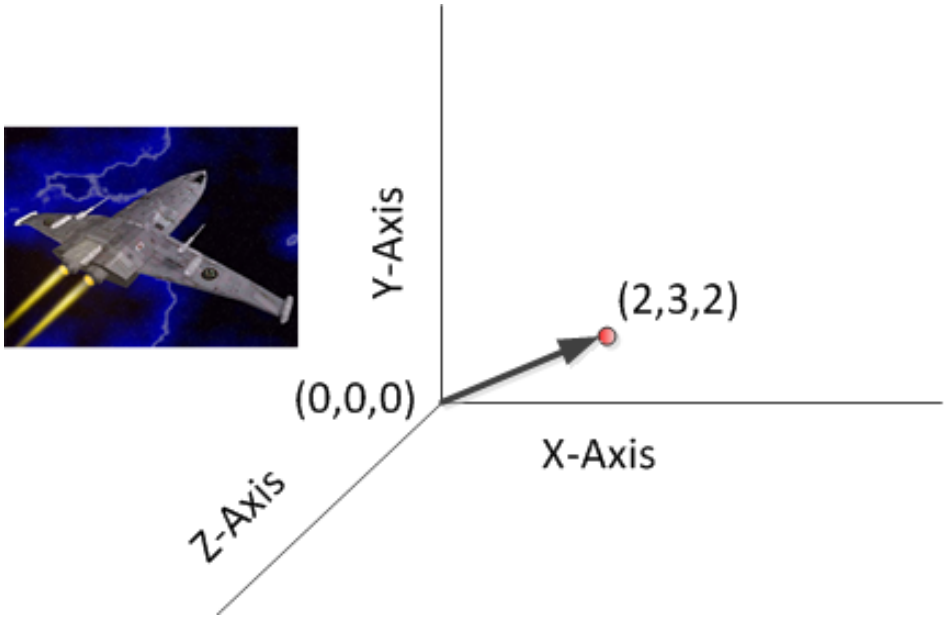
Reasoning about Indirect Effects

Indirect effects can be represented in static causal laws in $\mathcal{BC}+$:

- For example, Accelerating and decelerating not only affect the speed and the distance of the car, but also indirectly affect the speed and the distance of the bag in the car.

$$\textbf{caused } \mathit{Speed}(\mathit{Bag}) = x \textbf{ if } \mathit{Speed} = x \wedge \mathit{In}(\mathit{Bag}, \mathit{Car})$$
$$\textbf{caused } \mathit{Distance}(\mathit{Bag}) = x \textbf{ if } \mathit{Distance} = x \wedge \mathit{In}(\mathit{Bag}, \mathit{Car}) \ .$$

Reasoning about Additive Fluents



Describe the cumulative effects of firing multiple jets:

- In the language of CCALC:
  $Fire(j)$ **increments** $Vel(ax)$ **by** $n/Mass$ **if** $Force(j, ax) = n$
  limited to integer arithmetic.
- In $\mathcal{BC}+$:
  $Fire(j)$ **increments** $Vel(ax)$ **by** $n/Mass \times t$ **if** $Force(j, ax) = n \wedge Dur = t$.

Reasoning about Processes

The enhanced $\mathcal{BC}+$ is flexible enough to represent the start-process-end model, where instantaneous actions may initiate or terminate processes.

Example: Two Taps Water Tank with Leak
TurnOn(x) **causes** *On*$(x) \wedge$ *Dur* $= 0$
TurnOff(x) **causes** *On*$(x) = \text{FALSE} \wedge$ *Dur* $= 0$

On(x) **increments** *Level* **by** $\mathrm{W}(x) \times t$ **if** *Dur* $= t$
Leaking **increments** *Level* **by** $-(\mathrm{V} \times t)$ **if** *Dur* $= t$

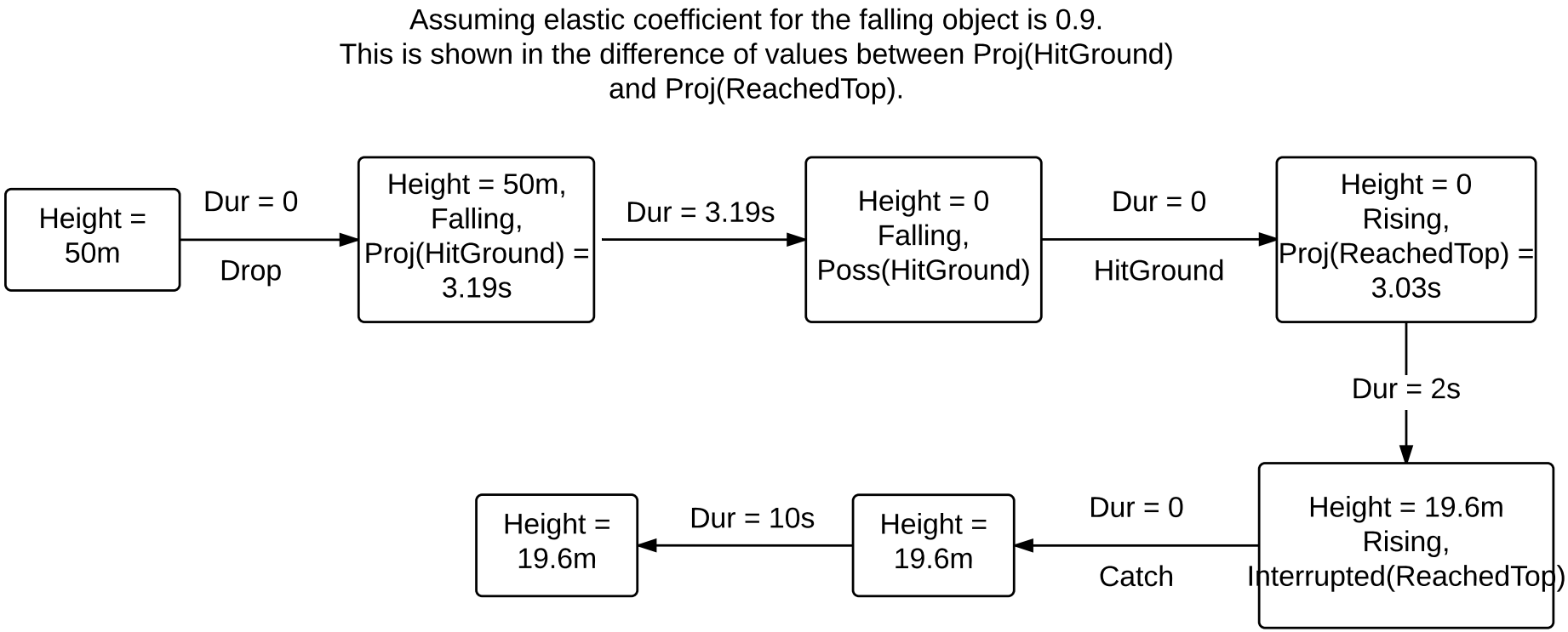
constraint $(\mathrm{Low} \leq \mathit{Level}) \wedge (\mathit{Level} \leq \mathrm{High})$
inertial *On*(x), *Leaking*
exogenous c for every action constant c

exogenous *Time*
constraint *Time* $= t + t'$ **after** *Time* $= t \wedge$ *Dur* $= t'$

Reasoning about Natural Actions

HitGround, ReachedTop are natural actions

Drop, Catch are agent's actions



Conclusion

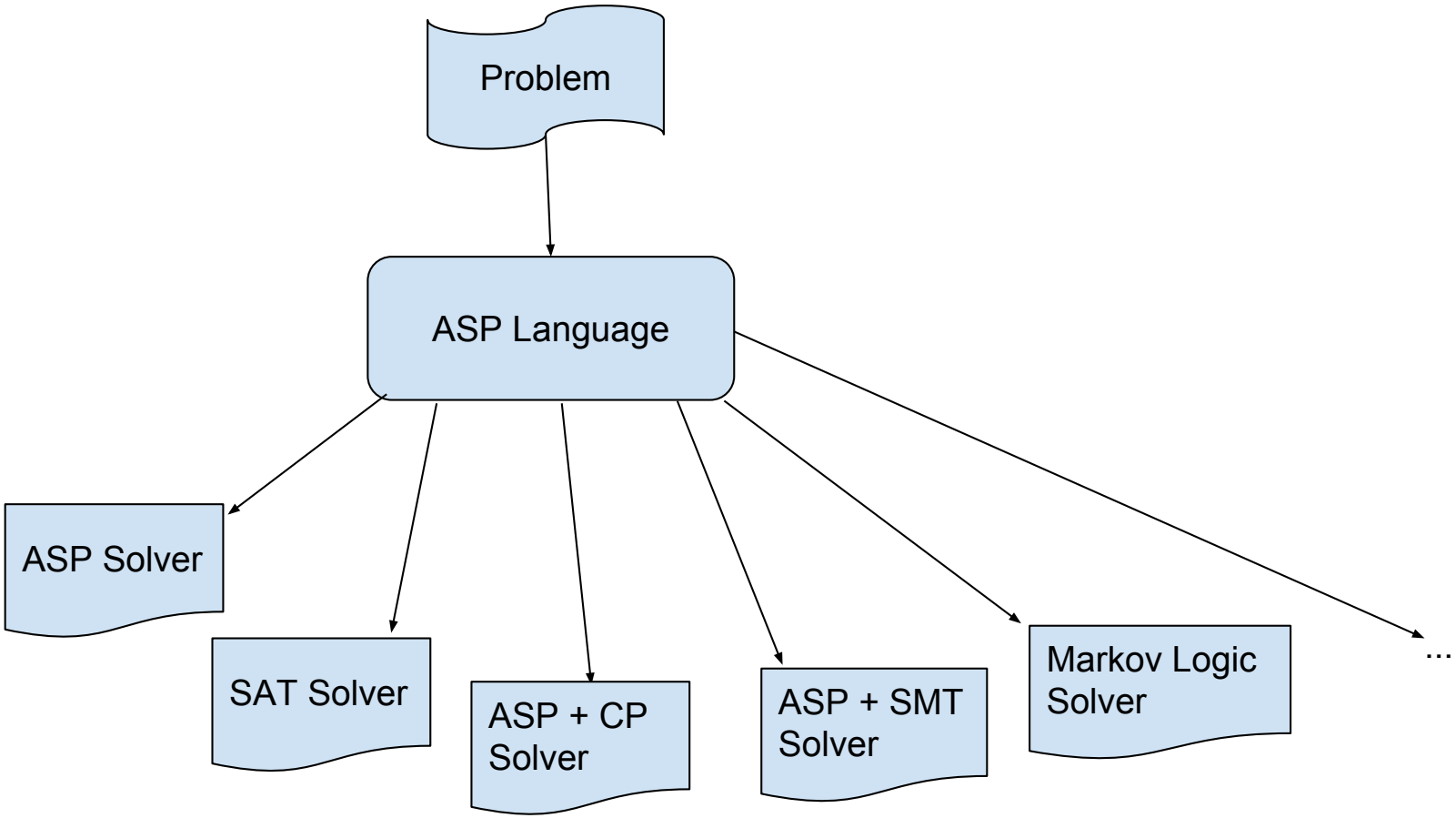
Conclusion

- ASPMT is a natural formalism that combines the advantages of ASP and SMT. Enhancements in ASP and SMT can be carried over to ASPMT.
- The language of ASPMT is based on functional stable model semantics, which can express default value assigned to functions. This feature makes possible a tight integration of ASP and other languages where functions are primitive constructs.
- We expect that many results known between ASP and SAT can be carried over to the relationship between ASPMT and SMT. Completion is one such example.
- The action language $\mathcal{BC}+$ defined by a reduction to ASPMT allows us to handle reasoning about hybrid systems, where discrete state changes and continuous changes coexist.

ASP as an Interface Language

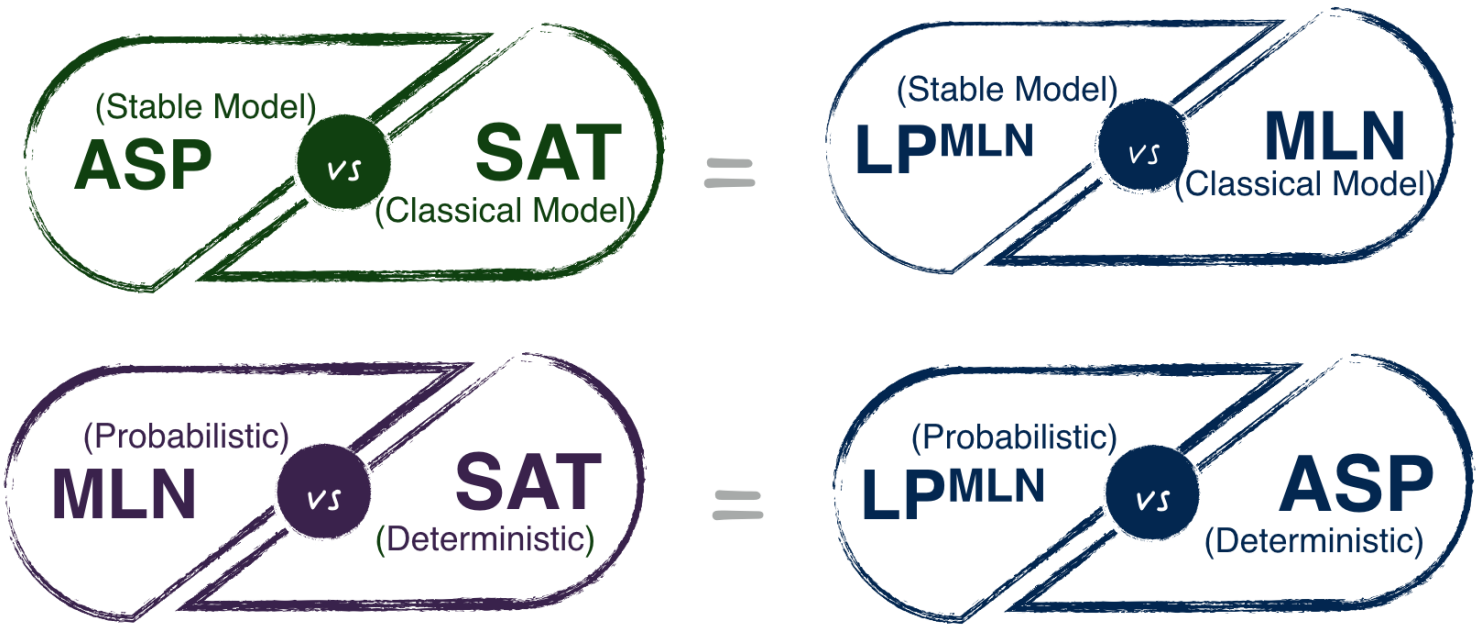
ASP language serve as a specification language for AI.

Computation is carried out by compilation to different engines.



Recent Work: Weighted Rules in ASP

- $\mathrm{LP}^{\mathrm{MLN}}$ [LW16] is an extension of ASP with weighted rules, similar to how Markov Logic [RD06] extends SAT/FOL.
- The weight of a "soft" stable model is determined by the weight of the rules that derive the stable model.
- $\mathrm{LP}^{\mathrm{MLN}}$ provides a way to comput ASP using statistical inference methods.



Conclusion

- ASP is an elegant knowledge specification language
  - allowing for various high level knowledge to be represented, while
  - computation can be carried out by different solvers/engines.
- First-order stable model semantics, taking into account default functions, provides a good ground for integrating ASP with other declarative paradigms.
- It also presents a simpler representation method in comparison with traditional ASP.
- In particular, high level action languages can be simply defined based on it.

Bibliography

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.

Marcello Balduccini.

Industrial-size scheduling with asp+cp.
In Proceedings of the 11th international conference on Logic programming and nonmonotonic reasoning, pages 284–296, 2011.

Chitta Baral, Michael Gelfond, and J. Nelson Rushton.

Probabilistic reasoning with answer sets.
TPLP, 9(1):57–144, 2009.

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.

Joseph Babb and Joohyung Lee.

Action language bc+: Preliminary report.
In Working Notes of the 7th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), 2014.

Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski.

Preferences and nonmonotonic reasoning.
AI Magazine, 29(4):69–78, 2008.

Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits.

Combining answer set programming with description logics for the semantic web.
Artificial Intelligence, 172(12-13):1495–1539, 2008.

Paolo Ferraris.

Answer sets for propositional theories.

In Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 119–131, 2005.

Paolo Ferraris and Vladimir Lifschitz.

On the stable model semantics of first-order formulas with aggregates.
In Proceedings of the 2010 Workshop on Nonmonotonic Reasoning, 2010.

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz.

Stable models and circumscription.
Artificial Intelligence, 175:236–263, 2011.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.

Recursive aggregates in disjunctive logic programs: Semantics and complexity.
In Proceedings of European Conference on Logics in Artificial Intelligence (JELIA), 2004.

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.

Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner.

Nonmonotonic causal theories.
Artificial Intelligence, 153(1–2):49–104, 2004.

Michael Gelfond, Vladimir Lifschitz, and Arkady Rabinov.

What are the limitations of the situation calculus?
In Robert Boyer, editor, Automated Reasoning: Essays in Honor of Woody Bledsoe, pages 167–179. Kluwer, 1991.

M. Gebser, M. Ostrowski, and T. Schaub.

Constraint answer set solving.
In Proceedings of International Conference on Logic Programming (ICLP), pages 235–249, 2009.

Tomi Janhunen, Guohua Liu, and Ilkka Niemelä.

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.

Henry Kautz and Bart Selman.

Planning as satisfiability.
In Proceedings of European Conference on Artificial Intelligence (ECAI), pages 359–363, 1992.

Vladimir Lifschitz.

Answer set programming and plan generation.
Artificial Intelligence, 138:39–54, 2002.

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.

Joohyung Lee, Vladimir Lifschitz, and Fangkai Yang.

Action language $\mathcal{BC}$: Preliminary report.
In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2013.

Joohyung Lee and Yunsong Meng.

On reductive semantics of aggregates in answer set programming.
In Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 182–195, 2009.

Joohyung Lee and Ravi Palla.

Integrating rules and ontologies in the first-order stable model semantics (preliminary report).
In Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 248–253, 2011.

Thomas Lukasiewicz.

Fuzzy description logic programs under the answer set semantics for the semantic web.
In Thomas Eiter, Enrico Franconi, Ralph Hodgson, and Susie Stephens, editors, RuleML, pages 89–96. IEEE Computer Society, 2006.

Joohyung Lee and Yi Wang.

Stable models of fuzzy propositional formulas.
In Proceedings of European Conference on Logics in Artificial Intelligence (JELIA), pages 326–339, 2014.


Joohyung Lee and Yi Wang.
Weighted rules under the stable model semantics.
In Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR), 2016.


Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe.
Well-founded and stable semantics of logic programs with aggregates.
TPLP, 7(3):301–353, 2007.


Matthew Richardson and Pedro Domingos.
Markov logic networks.
Machine Learning, 62(1-2):107–136, 2006.


Patrik Simons, Ilkka Niemelä, and Timo Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138:181–234, 2002.


Przemysław Andrzej Wałega, Mehul Bhatt, and Carl Schultz.
Aspmt (qs): non-monotonic spatial reasoning with answer set programming modulo theories.
In Logic Programming and Nonmonotonic Reasoning, pages 488–501. Springer, 2015.