Circumscriptive Event Calculus as Answer Set Programming

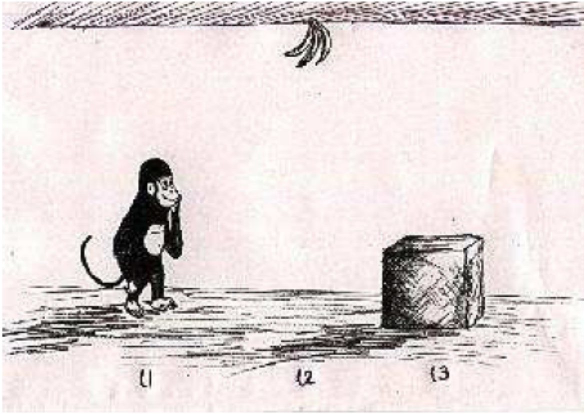
Joohyung Lee

Computer Science and Engineering
Arizona State University

Forum for AI
April 3, 2009

Reasoning about Actions

Concerned with developing appropriate systems of logic for describing the properties of actions.



fluent: anything that depends on the state of the world
action: anything that can be executed and may change the state of the world.

Challenges

- Frame Problem: how to formalize the commonsense law of inertia [McCarthy, 1969].
  - How do we describe things that remain unchanged by default?
- Qualification Problem: how to describe the conditions for actions to have intended effects.
- Ramification Problem: how to describe the indirect effects of actions [Finger, 1986].

Nonmonotonic Reasoning

- Human level intelligence requires defeasible reasoning: conclusions are drawn tentatively and can be retracted in the light of further information.
- Difficult to handle defeasible reasoning in first-order logic.
  - First-order logic is monotonic: if $\Gamma \vdash A$, then $\Gamma \cup \Delta \vdash A$.
  - Nonmonotonic formalisms: $\Gamma \vdash A$, but possibly $\Gamma \cup \Delta \nvdash A$.
    - completion [Clark, 1978]
    - circumscription [McCarthy, 1980]
    - default logic [Reiter, 1980]
    - a nonmonotonic logic [McDermott and Doyle, 1980]

Yale Shooting Problem [Hanks and McDermott, 1987]

Initially the turkey is alive and the gun is not loaded. Then the gun is loaded, and after wait, the gun is shot. Is the turkey alive?

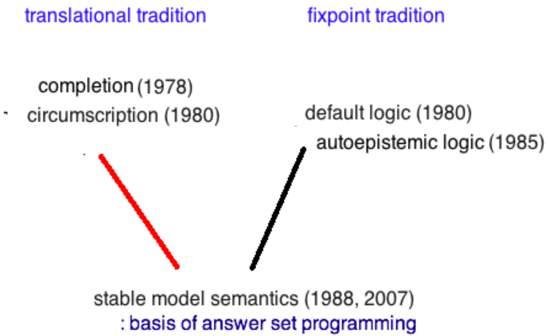
A naive attempt to formalize the commonsense law of inertia does not work.

It turns out that it is not the fault of nonmonotonic logics. Rather it is about how to use them in a proper way. More structured high level action formalisms are desirable.

Action Formalisms

Situation Calculus [McCarthy & Hayes, 1969] Event Calculus [Kowalski & Sergot, 1986] Temporal Action Logics [Doherty, 1996] ...	Action Languages $\mathcal{A}$ [Gelfond & Lifschitz, 1993] $\mathcal{C}$ [Giunchiglia & Lifschitz, 1998] $\mathcal{C}+$ [Giunchiglia *et al.*, 2004] ...
Circumscription [McCarthy, 1980;1986] Completion [Clark, 1978]	Stable Model Semantics [Gelfond & Lifschitz, 1988] Nonmonotonic Causal Theories [Giunchiglia *et al.*, 2004]
Classical Logic	Nonmonotonic Logics

This Talk is About ...



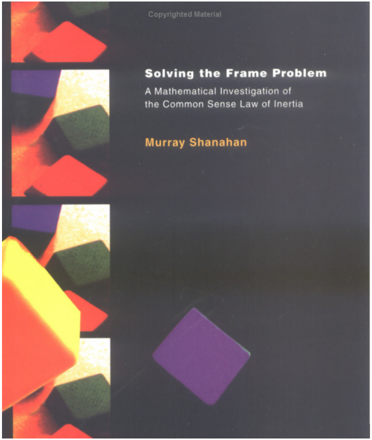
- Nonmonotonic logics are closely related to each other.
- Nonmonotonic logics are closely related to classical logic.
- Synergies can be obtained from the relationships.
- Circumscriptive event calculus can be reformulated in terms of the stable model semantics and can be computed by implementations of the stable model semantics.

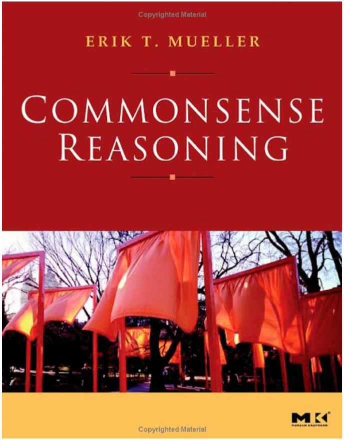
Contents

Brief History of Event Calculus

- Original version by Kowalski and Sergot, 1986: The theory can be represented as Horn clauses augmented by negation as failure, and can be run as a Prolog program.
- Extensions: [Kowalsky, 1986, 1992], [Eshghi, 1988], [Shanahan, 1988, 1989, 1990, 1997].
- Circumscriptive event calculus [Shanahan, 1995, 1997, 1998, 1999, 2004], [Miller and Shanahan, 1996], [Miller and Shanahan, 1999], [Mueller, 2004]: Extensive developments were carried out under the classical logic setting using circumscription.

Event Calculus





Shanahan, Solving the Frame Problem, 1997, MIT Press.
Mueller, Commonsense Reasoning, 2006, Morgan Kaufmann.

Event Calculus Predicates

Predicate	**Meaning**
$HoldsAt(f, t)$	f is true at t
$Happens(e, t)$	e occurs at t
$ReleasedAt(f, t)$	f is released from the commonsense law of inertia at t
$Initiates(e, f, t)$	if e occurs at t, then f is true and not released from the commonsense law of inertia after t
$Terminates(e, f, t)$	if e occurs at t, then f is false and not released from the commonsense law of inertia after t
$Releases(e, f, t)$	if e occurs at t, then f is released from the commonsense law of inertia after t
$Trajectory(f_1, t_1, f_2, t_2)$	if f_1 is initiated by an event that occurs at t_1, then f_2 is true at $t_1 + t_2$
$AntiTrajectory(f_1, t_1, f_2, t_2)$	if f_1 is terminated by an event that occurs at t_1, then f_2 is true at $t_1 + t_2$

Yale Shooting in Event Calculus

$$
\begin{aligned}
T: \quad & \mathit{Initiates}(\mathit{Load}, \mathit{Loaded}, t) \\
& \mathit{HoldsAt}(\mathit{Loaded}, t) \rightarrow \mathit{Terminates}(\mathit{Shoot}, \mathit{Alive}, t) \\
& \mathit{Terminates}(\mathit{Shoot}, \mathit{Loaded}, t) \\
& \\
& \mathit{HoldsAt}(\mathit{Alive}, 0) \\
& \neg \mathit{HoldsAt}(\mathit{Loaded}, 0) \\
& \mathit{Happens}(\mathit{Load}, 0) \\
& \neg \mathit{Happens}(e, 1) \\
& \mathit{Happens}(\mathit{Shoot}, 2)
\end{aligned}
$$

T does not entail $\neg \mathit{HoldsAt}(\mathit{Alive}, 3)$.

We need to add domain independent axioms in the Event Calculus.

We need to minimize the extents of *Initiates*, *Terminates*, *Happens*.

Domain Independent Event Calculus Axioms

$DEC9$:
$Happens(e,t) \land Initiates(e,f,t) \rightarrow HoldsAt(f,t{+}1)$

...

$DEC5$:
$$HoldsAt(f,t) \land \neg ReleasedAt(f,t{+}1) \land \\ \quad \neg \exists e(Happens(e,t) \land Terminates(e,f,t)) \rightarrow HoldsAt(f,t{+}1)$$

Circumscription [McCarthy, 1980, 1986]

$\mathrm{CIRC}[F; \mathbf{p}]$ is a second-order formula such that its models are the models of F that are minimal on $\mathbf{p}$.

$$\mathrm{CIRC}[F; \mathbf{p}] = F \wedge (\text{formula that makes } \mathbf{p} \text{ minimal})$$

- $\mathrm{CIRC}[p(a); p]$ is equivalent to

$$\forall x(p(x) \leftrightarrow x = a).$$

- $\mathrm{CIRC}[p(a) \wedge \forall x(p(x) \rightarrow q(x)); p, q]$ is equivalent to

$$\forall x(p(x) \leftrightarrow x = a) \wedge \forall (q(x) \leftrightarrow x = a).$$

Event Calculus Domain Description

An event calculus domain description is defined as

$$\mathrm{CIRC}[\Sigma;\ \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \ \mathrm{CIRC}[\Delta;\ \mathit{Happens}] \wedge F$$

where

- Σ is a conjunction of "effect" axioms
  - $[\mathit{condition}] \rightarrow \mathit{Initiates}(e, f, t)$
  - $[\mathit{condition}] \rightarrow \mathit{Terminates}(e, f, t)$
  - $[\mathit{condition}] \rightarrow \mathit{Releases}(e, f, t)$
- Δ is a conjunction of "event occurrence" axioms
  - $\mathit{Happens}(e, t)$
- F is a conjunction of first-order sentences describing UNA, observations and the *event calculus axioms*.

Computing Event Calculus

In general, circumscription is not reducible to first-order logic.

Approximation: completion [Clark, 1978]. Under certain conditions circumscription coincides with completion [Lifschitz, 1994].

Completion turns if conditions into if and only if conditions.

$$\begin{gathered} \mathit{Raining} \rightarrow \mathit{Wet} \\ \mathit{SprinklerOn} \rightarrow \mathit{Wet} \end{gathered}$$

Completion, applied to Wet, yields

$$\mathit{Wet} \leftrightarrow \mathit{Raining} \vee \mathit{SprinklerOn}.$$

Circumscription entails completion, but not the other way around. For instance, completion turns $\mathit{Wet} \rightarrow \mathit{Wet}$ into $\mathit{Wet} \leftrightarrow \mathit{Wet}$.

Event Calculus Reasoning Tools

- Prolog
- Abductive logic programming [Eshghi, 1988; Shanahan, 1989]
- SAT-based approach [Shanahan & Witkowski, 2004; Mueller, 2004]
  - DEC reasoner: reduces event calculus reasoning to satisfiability checking by turning circumscription into completion, and then using SAT solvers.
- ASP-based approach
  - ECASP: reduces EC to ASP.

Yale Shooting in the DEC Reasoner

```
[time] Initiates(Load(),Loaded(),time).
[time] HoldsAt(Loaded(),time)
           -> Terminates(Shoot(),Alive(),time).
[time] Terminates(Shoot(),Loaded(),time).

HoldsAt(Alive(),0).
!HoldsAt(Loaded(),0).
Happens(Load(),0).
[event] !Happens(event,1).
Happens(Shoot(),2).
```

Yale Shooting in the DEC Reasoner: Output

```
Discrete Event Calculus Reasoner 1.0
loading yale.e
loading foundations/Root.e
loading foundations/EC.e
24 variables and 60 clauses
relsat solver
1 model
---
model 1:
0
Alive().
Happens(Load(), 0).
1
+Loaded().
2
Happens(Shoot(), 2).
3
-Alive().
-Loaded().
...
encoding 0.1s
solution 0.0s
total 0.3s
```

Introduction to Answer Set Programming

What is Answer Set Programming (ASP)?

A new form of declarative programming oriented towards combinatorial search problems and knowledge-intensive applications.

The idea of ASP is to represent a given search problem as the problem of finding an answer set for some logic program, and then find a solution using an answer set solver.

ASP Example: 8-Queens Problem

```
num(1..8).
#domain num(I;I1;J;J1).

1 {q(I,J): num(J)} 1.
:- q(I,J), q(I1,J), I<I1.
:- q(I,J), q(I1,J1), I<I1, I1-I==abs(J1-J).
```

Given the input, SMODELS returns 92 answer sets, which correspond 1-1 with 92 solutions.

Brief History of Answer Set Programming

1988: Definition of answer sets for Prolog-like programs.
1992: Extending the definition to more general programs.
1996: SMODELS: first answer set solver.
1999: ASP identified as a new programming paradigm.

ASP has been applied to various knowledge-intensive tasks, such as knowledge representation, planning, diagnosis, decision support systems, model checking, production configuration, VLSI routing, the Semantic Web, computational linguistics, bioinformatics, ...

WASP (Working Group on Answer Set Programming) : 17 European universities in 8 countries. Funded by EU.

conferences/workshops : LPNMR, ASP, ASPOCP, laSh

Biennial ASP solver competition.

ASP vs. SAT

Like SAT, ASP provides a common basis for formalizing and solving various problems. On the other hand, ASP focuses on knowledge representation.

Some ASP solvers use SAT solvers as search engines, e.g., CMODELS (Lierler).

Paradigm	SAT	ASP
Input	set of clauses	set of rules
Solutions	models	stable models (answer sets)
	monotonic	nonmonotonic
Solvers	MiniSAT, zChaff, Jerusat, Walksat, Relsat...	Smodels, NoMore, DLV, ASSAT, Cmodels, SAG, clasp, ...

Stable Model Semantics (a.k.a. Answer Set Semantics)

A nonmonotonic formalism. Mathematical basis of answer set programming.

Started as a theory to explain the meaning of negation as failure in Prolog.

$$A_0 \quad \leftarrow \quad A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$$

means intuitively that

If you have generated $A_1, \ldots, A_m$, and

it is impossible to generate any of $A_{m+1}, \ldots, A_n$,

then you may derive A_0.

Formal Definition

X is a stable model of Π if X is the smallest set of atoms satisfying Π^X.

Example: Π:

$$p \leftarrow not\ q$$
$$q \leftarrow not\ p$$

X	$\{p\}$	$\{p, q\}$
Π^X	$p \leftarrow \top$ $q \leftarrow \bot$	$p \leftarrow \bot$ $q \leftarrow \bot$
	$\{p\}$ is an answer set	$\{p, q\}$ is not an answer set

Can be viewed as a special case of Reiter's default logic.

The theory has been extended to allow various constructs, disjunctions in the head, choice rules, aggregates, constraints, preferences.

Stable Model Semantics and Propositional Logic

Embedding propositional logic into the stable model semantics is straightforward.

The other direction is non-trivial, but possible.

Theorem on Loop Formulas The stable models of Π are exactly the models of Π that satisfy all loop formulas.

Π_1	Π_1	$\cup$	loop formulas
$p \leftarrow not\ s$	$\neg s \rightarrow p$		$p \rightarrow \neg s \vee r$
$p \leftarrow r$	$r \rightarrow p$		$q \rightarrow r$
$q \leftarrow r$	$r \rightarrow q$		$r \rightarrow p \wedge q$
$r \leftarrow p, q$	$p \wedge q \rightarrow r$		$s \rightarrow \bot$
			$p \wedge r \rightarrow \neg s$
			$q \wedge r \rightarrow \bot$
			$p \wedge q \wedge r \rightarrow \neg s$

Π_1 has six models: $\{p\}$, $\{s\}$, $\{p, s\}$, $\{q, s\}$, $\{p, q, r\}$, $\{p, q, r, s\}$. Only $\{p\}$ is stable, and is the only model that satisfies all loop formulas.

Theorem on Loop Formulas

Some nonmonotonic logics can be turned into propositional logic

- Answer sets = Propositional Logic (PL) representation + loop formulas [Lin & Zhao, AAAI 2002], [ICLP 2003].
- Circumscription = PL representation + loop formulas [AAAI 2004]
- Causal logic = PL representation + loop formulas [LPNMR 2004]
- Completion is a special case of loop formulas [IJCAI 2005]
- Generalization to arbitrary propositional formulas under the stable model semantics [AMAI 2006]
- Refinement by elementary sets [AAAI 2006]

SAT-based ASP: SAT solvers are used for computing answer sets.

Get the best of possible worlds: expressive nonmonotonic languages can be computed using efficient SAT solvers.

New Definition of Stable Models

ASP is based on Grounding

Variables in ASP are understood in terms of grounding.

$$
\begin{array}{l}
p(a) \\
q(b) \\
r(x) \leftarrow p(x), not\ q(x)
\end{array}
$$

is shorthand for the formula

$$
\begin{array}{l}
p(a) \\
q(b) \\
r(a) \leftarrow p(a), not\ q(a) \\
r(b) \leftarrow p(b), not\ q(b).
\end{array}
$$

Grounding is required for applying fixpoint definition.

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

Needs to go beyond Grounding

Monotonic	Nonmonotonic
Propositional logic	ASP with no variables
First-order logic	ASP with variables (?)

Grounding often leads to semantic and computational limitations.

- Considerable time and memory requirements when variables range over a huge domain.
- Requires complete knowledge about the domain to be modelled.
  - Cannot handle open domains, where not every object is known in advance;
  - Cannot handle dynamically changing domains, where objects are being created and destroyed.
- Grounding destroys the structure of the first-order theory.

New Definition of Stable Models

[Ferraris, Lee and Lifschitz, IJCAI 2007].

A new answer for "what is a stable model?" "how do we avoid grounding in the stable model semantics?"

- Idea 1: Treat ASP programs as alternative notation for first-order formulas.

Logic program	FOL-representation
$p(a)$	$p(a) \wedge q(b) \wedge$
$q(b)$	$\forall x(p(x) \wedge \neg q(x) \rightarrow r(x) \vee s(x))$
$s(x); r(x) \leftarrow p(x), \textit{not}\ q(x)$	

- Idea 2: Define the stable models of a formula F as the models that satisfy the "stability" condition.

$$\mathrm{SM}[F; \mathbf{p}] = F \wedge \textit{Kinds of Loop Formulas}$$

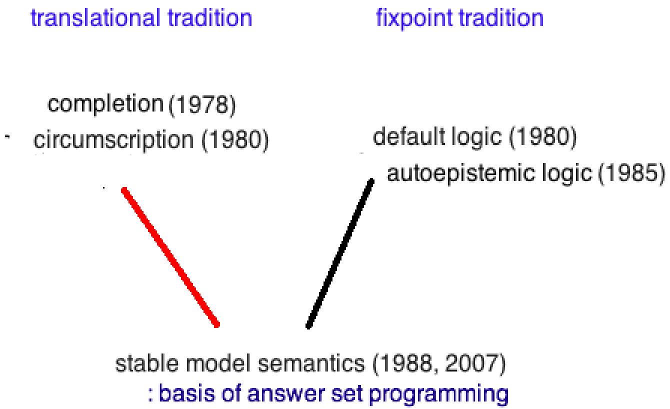
Similar to circumscripton. (c.f. stability vs. minimality)

Overcoming the Mismatch

	FOL	ASP	New Language
Syntax	1st-order formulas	rules	1st-order formulas
	$\forall$, $\exists$	no $\forall$, no $\exists$	$\forall$, $\exists$
Semantics	monotonic	nonmonotonic	nonmonotonic
	models	stable models	stable models
	no negation as failure	negation as failure (*not*)	negation as failure ($\neg$)
	genuine variables	schematic variables	genuine variables
	no built-in UNA, DCA	built-in UNA, DCA	no built-in UNA, DCA
		fixpoint definition	in terms of 2nd-order logic

Both FOL and ASP can be embedded into the new language of stable models.

Circumscription and Stable Models



- the stable model semantics can be turned into circumscription [Ferraris, Lee and Lifshitz, IJCAI 2007].
- circumscription can be turned into the stable model semantics [Lee and Lin, AIJ 2006], [Kim, Lee and Palla, IJCAI 2009].

Event Calculus as Answer Set Programming

situation calculus circumscriptive event calculus temporal action logics ...	Action languages $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{C}+$ ...
circumscription completion	stable model semantics nonmonotonic causal theories
classical logic	nonmonotonic logics

How are action formalims based on classical logic and the stable model semantics are related to each other?

- Circumscriptive Event Calculus can be reformulated in terms of the stable model semantics [Kim, Lee and Palla, IJCAI 2009].
- ASP solvers can be applied to computing event calculus, and this approach outperforms the state-of-the-art SAT-based Event Calculus reasoning tools.

Circumscription and Stable Models

Theorem For any canonical theory F relative to $\mathbf{p}$,

$$\mathrm{CIRC}[F; \mathbf{p}] \Leftrightarrow \mathrm{SM}[F; \mathbf{p}].$$

Example:

$$F = p(x) \wedge \neg\exists y(q(x, y) \wedge r(x, y)) \rightarrow s(x).$$

is canonical w.r.t. $\{p, s\}$.
Therefore $\mathrm{CIRC}[F; p, s] \Leftrightarrow \mathrm{SM}[F; p, s]$.

Turning Event Calculus Description to SM

Theorem Let $\mathbf{p}$ be the set of all predicates (other than equality and comparisons) occurring in the event calculus description. The following theories are equivalent:

(a) $\mathrm{CIRC}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{CIRC}[\Delta; \mathit{Happens}] \wedge F$

(b) $\mathrm{SM}[\Sigma; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}] \wedge \mathrm{SM}[\Delta; \mathit{Happens}] \wedge F$

(c) $\mathrm{SM}[\Sigma \wedge \Delta \wedge F; \mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}]$

(d) $\mathrm{SM}[\Sigma \wedge \Delta \wedge F \wedge \mathit{Choice}(\mathbf{p} \setminus \{\mathit{Initiates}, \mathit{Terminates}, \mathit{Releases}, \mathit{Happens}\})]$

$\mathit{Choice}(\mathbf{q})$ denotes the conjunction of "choice formulas" $\forall \mathbf{x}(q(\mathbf{x}) \vee \neg q(\mathbf{x}))$ for all predicate constants q in $\mathbf{q}$.

Turning Event Calculus Description to ASP

$$(HoldsAt(f,t) \land \neg ReleasedAt(f,t{+}1) \land \\ \neg \exists e(Happens(e,t) \land Terminates(e,f,t))) \rightarrow HoldsAt(f,t{+}1).$$

is turned into the conjunction of

$$(HoldsAt(f,t) \land \neg ReleasedAt(f,t+1) \land \\ \neg q(f,t)) \rightarrow HoldsAt(f,t+1)$$
$$Happens(e,t) \land Terminates(e,f,t) \rightarrow q(f,t)$$

and then turned into rules

$$HoldsAt(f,t{+}1) \leftarrow HoldsAt(f,t), not\ ReleasedAt(f,t{+}1), \\ not\ q(f,t))$$
$$q(f,t) \leftarrow Happens(e,t),\ Terminates(e,f,t)$$

ECASP vs. DEC reasoner

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

http://decreasoner.sourceforge.net/csr/ecas/

ASP-based vs. SAT-based Approach

- DEC reasoner is based on the reduction of circumscription to completion. Able to solve 11 out of 14 benchmark problems.

- ECASP can handle the *full* version of the event calculus (modulo grounding). Able to solve all 14 problems.
- For example, the following axiom cannot be handled by the DEC reasoner, but can be done by the ASP approach.

$$\begin{array}{l} HoldsAt(HasBananas, t) \\ \wedge Initiates(e, At(Monkey, l), t) \rightarrow Initiates(e, At(Bananas, l), t) \end{array}$$

- ECASP computes faster.

Experiments (I)

Problem (max. time)	DEC reasoner	ECASP w/ LPARSE + CMODELS	ECASP w/ GRINGO + CLASP	ECASP w/ CLINGO
BusRide (15)	—	0.48 (0.42+0.06) A:156/R:7899/C:188	0.04 (0.03+0.01) A:733/R:3428	—
Commuter (15)	—	498.11 (447.50+50.61) A:4913/R:7383943/C:4952	44.42 (37.86 + 6.56) A:24698/R:5381620	28.79
Kitchen Sink (25)	71.10 (70.70+0.40) A:1014/C:12109	43.17 (37.17+6.00) A:123452/R:482018/C:0	2.47 (1.72+0.75) A:114968/R:179195	2.03
Thielscher Circuit (20)	13.9 (13.6+0.3) A:5138/C:16122	0.42 (0.38+0.04) A:3160/R:9131/C:0	0.07 (0.05+0.02) A:1686/R:6510	0.05
Walking Turkey (15)	—	0.05 (0.04+0.01) A:556/R:701/C:0	0.04 (0.01+0.03) A:364/R:503	0.01

A: number of atoms, C: number of clauses, R: number of ground rules

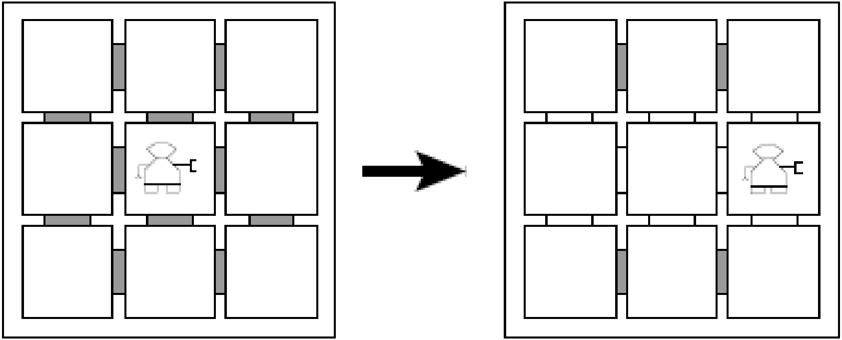
DEC reasoner and CMODELS used the same SAT solver RELSAT.

Experiments (II)

Problem (max. time)	DEC reasoner	ECASP w/ LPARSE + CMODELS	ECASP w/ GRINGO + CLASP	ECASP w/ CLINGO
Falling w/ AntiTraj (15)	270.2 (269.3+0.9) A:416/C:3056	0.74 (0.66+0.08) A:5757/R:10480/C:0	0.10 (0.08+0.02) A:4121/R:7820	0.08
Falling w/ Events (25)	107.70 (107.50+0.20) A:1092/C:12351	34.77 (30.99+3.78) A:1197/R:390319/C:1393	2.90 (2.01+0.89) A:139995/R:208282	2.32
HotAir Baloon (15)	61.10 (61.10+0.00) A:288/C:1163	0.19 (0.16+0.03) A:489/R:2958/C:678	0.04 (0.03+0.01) A:1137/R:1909	0.03
Telephone1 (40)	18.00 (17.50+0.50) A:5419/C:41750	1.70 (1.51+0.19) A:23978/R:30005/C:0	0.31 (0.26+0.05) A:21333/R:27201	0.25

A: number of atoms, C: number of clauses, R: number of ground rules

Hybrid of ASP and EC



How to open doors so that every room is accessible from each other?

The goal cannot be represented in the Event Calculus.

Robby in ECASP

```
[room,time] Initiates(Go(room),InRoom(room),time).

[room,room1,time] (HoldsAt(InRoom(room1),time)
        -> Terminates(Go(room),InRoom(room1),time)).

[room,time] (Happens(Go(room),time)
  -> {door,room1}(Sides(room,room1,door) &
                !HoldsAt(Locked(door),time) &
                 HoldsAt(InRoom(room1),time))).


...

_asp {
  accessible(Room1,Room2,Time) :- not holdsAt(locked(Door),Time),
                                  sides(Room1,Room2,Door).

  accessible(Room,Room2,Time) :- accessible(Room,Room1,Time).
                                 accessible(Room1,Room2,Time).
}.
```

Conclusion

- The new language of stable models is a suitable nonmonotonic formalism as general as circumscription, with the unique advantage of having efficient ASP solvers as computational tools.
- ASP solvers can be used as a general reasoning engine for circumscription based approaches, such as circumscriptive event calculus. This approach can handle the *full* version of the event calculus, modulo grounding.
- Nonmonotonic logics and classical logic are closely related to each other. Synergies can be obtained by combining them.

Circumscription

The circumscription of a first-order sentence F over predicate constants $\mathbf{p} = (p_1, \ldots, p_n)$ of F is the second-order formula

$$\mathrm{CIRC}[F; \mathbf{p}] = F \wedge \neg \exists \mathbf{u}(\mathbf{u} < \mathbf{p} \wedge F(\mathbf{u}))$$

where

- $\mathbf{u}$ is a list of predicate variables similar to $\mathbf{p}$;
- $\mathbf{u} < \mathbf{p}$ stands for a formula that expresses that $\mathbf{u}$ is "strictly stronger" than $\mathbf{p}$.
  e.g. $u < p = \forall x(u(x) \rightarrow p(x)) \wedge \neg \forall x(p(x) \rightarrow u(x))$
- $F(\mathbf{u})$ stands for F with each predicate constant in $\mathbf{p}$ replaced by the corresponding variable in $\mathbf{u}$.

A model of $\mathrm{CIRC}[F; \mathbf{p}]$ is a model of F that is minimal on $\mathbf{p}$.

Example

$$\mathrm{CIRC}[F;\mathbf{p}] = F \wedge \neg\exists\mathbf{u}(\mathbf{u} < \mathbf{p} \wedge F(\mathbf{u}))$$

For $F = p(a) \wedge \forall x(p(x) \rightarrow q(x))$,

$$\begin{aligned}\mathrm{CIRC}[F;p,q] = {} & p(a) \wedge \forall x(p(x) \rightarrow q(x)) \\ & \wedge \neg\exists uv(((u,v) < (p,q)) \\ & \quad \wedge u(a) \wedge \forall x((u(x) \rightarrow v(x)))) \\ & \leftrightarrow \forall x(p(x) \leftrightarrow x = a) \wedge \forall x(p(x) \leftrightarrow q(x)).\end{aligned}$$

$$\begin{aligned}(u,v) < (p,q) = {} & \forall\mathbf{xy}(((u(\mathbf{x}) \rightarrow p(\mathbf{x})) \wedge (v(\mathbf{y}) \rightarrow q(\mathbf{y})))) \wedge \\ & \neg\forall\mathbf{xy}((p(\mathbf{x}) \rightarrow u(\mathbf{x})) \wedge (q(\mathbf{y}) \rightarrow v(\mathbf{y})))\end{aligned}$$

Formal Definition

$$A_0 \quad \leftarrow \quad A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$$

The reduct Π^X is obtained from Π by replacing every occurrence of $\mathit{not}\ A$ by $\top$ if $X \models \mathit{not}\ A$ and $\bot$ otherwise.

X is a stable model of Π if X is the smallest set of atoms satisfying Π^X.

Example: Π:

$$p \leftarrow \mathit{not}\ q$$
$$q \leftarrow \mathit{not}\ p$$

X	$\{p\}$	$\{p, q\}$
Π^X	$p \leftarrow \top$ $q \leftarrow \bot$	$p \leftarrow \bot$ $q \leftarrow \bot$
	$\{p\}$ is an answer set	$\{p, q\}$ is not an answer set

We say that an occurrence of a predicate constant in a formula F is strictly positive if the occurrence is not in the antecedent of any implication.

We call implication $G \rightarrow H$ canonical w.r.t. $\mathbf{p}$ if in each of G and H, every occurrence of predicate constants from $\mathbf{p}$ is strictly positive. For instance,

$$F_1 = p(x) \wedge \neg\exists y(q(x,y) \wedge r(x,y)) \rightarrow s(x).$$

is canonical relative to $\{p, s\}$.

Theorem Let F be the universal closure of a conjunction of canonical implications w.r.t. $\mathbf{p}$. Then

$$\mathrm{SM}[F; \mathbf{p}] \Leftrightarrow \mathrm{CIRC}[F; \mathbf{p}].$$

For instance, $\mathrm{SM}[F_1; p, s] \Leftrightarrow \mathrm{CIRC}[F_1; p, s]$.

New Definition of Stable Models

The stable models of a first-order sentence F relative to intensional predicates $\mathbf{p}$ are the models of the second-order formula

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

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

- $p_i(t_1,\dots,t_m)^* = \begin{cases} u_i(t_1,\dots,t_m) & \text{if } p_i \text{ belongs to } \mathbf{p}, \\ p_i(t_1,\dots,t_m) & \text{otherwise;} \end{cases}$
- $(t_1 = t_2)^* = (t_1 = t_2)$;
- $\bot^* = \bot$;
- $(F \odot G)^* = F^* \odot G^* \qquad (\odot \in \{\wedge, \vee\})$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(QxF)^* = QxF^* \qquad (Q \in \{\forall, \exists\})$.

($\neg F$ is shorthand for $F \rightarrow \bot$.)

Relationship with the Originial Definition

ASP programs are a special case of the new language.

Theorem Given a program Π, a set of ground atoms is a stable model of Π under the 1988 definition iff it is a Herbrand stable model of Π under the new definition.

Example: $\{p(a), q(a)\}$ is the unique

- stable model of $\begin{cases} p(a) \\ q(x) \leftarrow p(x), \textit{not}\ r(x) \end{cases}$
  under the 1988 definition.

- Herbrand stable model of $p(a) \wedge \forall x(p(x) \wedge \neg r(x) \rightarrow q(x))$
  under the new definition.