Agent Logic Programs — A Declarative Agent Programming Language Based on Action Theories

Conrad Drescher, Stephan Schiffel, Michael Thielscher

Technische Universität Dresden

18.09.2009

Highlevel Agent Control Languages

... express

- agent control strategies and
- planning heuristics

wrt. a (formal) world model

Highlevel Agent Control Languages II

Prominent examples

- Golog (with Situation Calculus world model)
- Flux (with Fluent Calculus world model)
- Language by Witkowski and Shanahan (with Event Calculus world model)
- AgentSpeak etc. (no logic-based world model)

Highlevel Agent Control Languages III

These languages are

- \+ very expressive
- \+ efficiently implemented
- – tied to specific action calculi
- – more or less procedural
- – hard to provide with a declarative semantics
  (use computation tree semantics, transition semantics)

Agent Logic Programs

- combine definite logic programs and action calculi
- work with various action calculi
- have plain first order semantics

Action Calculus

Fundamental Ideas of Action Calculi

Properties can be

- fluent (not rigid)
- non-fluent (rigid)

An Axiomatization features

- action preconditions
- action effects
- initial state
- domain constraints (always true)

ALPs use Unifying Action Calculus (UAC)

- based on many-sorted first order logic with equality
- reifies fluents $\Rightarrow$ first order quantification over fluents
- standard predicates:
  - Poss : ACTION $\times$ TIME $\times$ TIME, and
  - Holds : FLUENT $\times$ TIME
- can be instantiated by
  - Event, Fluent, Situation Calculus
  - Planning calculi (ADL, ...)
  - ...

Blocksworld in the UAC

Action preconditions:

$$
\begin{aligned}
&(\forall)\mathrm{Poss}(\mathrm{Move}(\mathrm{block}_1, x, y), s_1, s_2) \equiv \\
&\quad \mathrm{Holds}(\mathrm{On}(\mathrm{block}_1, x), s_1) \wedge x \neq y \wedge \\
&\quad \neg(\exists \mathrm{block}_2)\mathrm{Holds}(\mathrm{On}(\mathrm{block}_2, \mathrm{block}_1), s_1) \wedge \\
&\quad (\neg(\exists \mathrm{block}_3)\mathrm{Holds}(\mathrm{On}(\mathrm{block}_3, y), s_1) \vee y = \mathrm{Table}) \wedge \\
&\quad s_2 = \mathrm{Do}(\mathrm{Move}(\mathrm{block}_1, x, y), s_1)
\end{aligned}
$$

Blocksworld in the UAC II

Action effects:

$$(\forall)\mathrm{Poss}(\mathrm{Move}(\mathrm{block}_1, x, y), s_1, s_2) \supset$$
$$[(\forall f)((\mathrm{Holds}(f, s_1) \wedge f \neq \mathrm{On}(\mathrm{block}_1, x)) \vee f = \mathrm{On}(\mathrm{block}_1, y))$$
$$\equiv \quad \mathrm{Holds}(f, s_2)]$$

Blocksworld in the UAC III

Domain constraints:

$$(\forall s, x)(x \neq \text{Table} \supset (\exists! y)\text{Holds}(\text{On}(x, y), s)$$

Initial state:

$$\text{Holds}(\text{On}(\text{Block}_1, \text{Table}), S_0) \wedge \text{Holds}(\text{On}(\text{Block}_2, \text{Block}_1), S_0)$$

Time Structures

For Agent Logic Programs we admit

- the natural numbers,
- the positive real numbers,
- situation terms,

$\Rightarrow$ We have branching and linear time

Agent Logic Programs

Basic idea of Agent Logic Programs (ALPs)

The action theory Σ specifies all possible behaviours

The program Π uses rules (i.e. Horn clauses) to select the ”good” behaviours

Agent Logic Programs

ALPs are definite programs with two special atoms:

- `do(A)`, for executing an action
- `?(Phi)`, for querying a state property ϕ

Example ALP for the blocks world:

```
strategy :-
  ?(forall(X,(on(X,table) v X=table))).
strategy :-
  do(move(Block,X,table)), strategy.
```

Macro-expanding ALPs

ALPs do not refer to the underlying time structure TIME

ALP rules are (temporally ordered) sequences

- Macro-expand to first-order clauses (unordered): extend literals by two arguments of sort TIME, prior/after
- `do(A)` is expanded to Poss(a, s_1, s_2)
- `?(Phi)` is expanded to HOLDS(ϕ, s)

Temporalized ALP

Example Macro-expanded ALP for the blocks world:

$$(\forall)\text{Strategy}(s, s) \subset$$
$$\quad \text{HOLDS}((\forall x)\text{On}(x, \text{Table}) \vee x = \text{Table}, s)$$
$$(\forall)\text{Strategy}(s_1, s_3) \subset$$
$$\quad \text{Poss}(\text{Move}(\text{block}, x, \text{Table}), s_1, s_2) \wedge \text{Strategy}(s_2, s_3)$$

The query `?- strategy.` is expanded to $(\exists s)\text{Strategy}(S_0, s)$

HOLDS-macro for State Properties

In ALPs we write e.g.

```
?(forall(X,(on(X,table) v X=table)))
```

This is macro-expanded to
$\mathrm{HOLDS}((\forall x)\mathrm{On}(x, \mathrm{Table}) \vee x = \mathrm{Table}, s)$

This corresponds to
$((\forall x)\mathrm{Holds}(\mathrm{On}(x, \mathrm{Table}), s) \vee x = \mathrm{Table})$

We treat HOLDS as atomic

$\Rightarrow$ Macro-expanded ALPs are definite logic programs

Semantics of Agent Logic Programs

What have we got?

- Expanded ALPs are sets of first order clauses
- Background theories are first order, too

What do we get?

- ALPs admit plain first order semantics

Proof Calculi for ALPs

Proof Calculi for ALPs

- based on SLD-resolution (use Prolog for implementation)
- assume UNA — avoid equational unification
- assume reasoning about Σ is sound and complete

The Problematic General Case

In LP, if $\models (\exists) G(\vec{x})$ then there is a substitution θ st. $\models (\forall) G(\vec{x})\theta$

The Problematic General Case

In LP, if $\models (\exists) G(\vec{x})$ then there is a substitution θ st. $\models (\forall) G(\vec{x})\theta$

But: Assume we have disjunctive knowledge

$$\text{Holds}(\text{On}(\text{Block}_1, \text{Table}), S_0) \vee \text{Holds}(\text{On}(\text{Block}_2, \text{Table}), S_0)$$

or existential knowledge

$$(\exists x)\text{Holds}(\text{On}(x, \text{Table}), S_0)$$

For the query $(\exists x)\text{Holds}(\text{On}(x, \text{Table}), S_0)$ there is no suitable substitution

Proof Calculus I — Plain SLD Resolution

- We disallow existential and disjunctive information
- Then SLD resolution is sound and complete

Addressing the General Case

Problem

We want to represent disjunctive or existential information

Solution

We can do this if we use Constraint Logic Programming

Constraint Logic Programming

Constraint Logic Programming over $\mathcal{X}$ (CLP($\mathcal{X}$)) is a scheme:

- CLP($\mathcal{X}$) extends LP by special atoms C, the *constraints*
- Constraints C are evaluated against first order constraint theory $\mathcal{X}$

CLP($\mathcal{X}$) Proof Calculus

The rule for constraints:

$$\frac{< (\neg G_1 \vee \ldots \vee \neg C \vee \ldots \vee \neg G_n),\ \mathcal{S} >}{< (\neg G_1 \vee \ldots \vee \neg G_n),\ \mathcal{S}' >}$$

where C is a constraint and $\mathcal{X} \models (\forall) C \wedge \mathcal{S} \equiv \mathcal{S}'$

Basic Ideas of CLP(Σ)

Constraint programming over action domains Σ:

- Action domain Σ as constraint domain $\mathcal{X}$
- The constraints C are $\mathrm{Poss}(a, s_1, s_2)$ and $\mathrm{HOLDS}(\varphi, s)$

In general Σ can be an arbitrary FO theory — our calculus works for arbitrary FO theories

Disjunctive Substitution

Ordinary Substitutions:

$$x = A \wedge y = B$$

Disjunctive Substitutions:

$$(x = A \wedge y = B) \vee (x = C \wedge y = D)$$

Denote such DNF formulas by Θ

Inference for CLP(Σ)

CLP(Σ) uses

- disjunctive substitutions for disjunctive information (we have to do reasoning by cases because of this)
- adds existential information to the constraint store

Soundness of CLP(Σ)

The results for CLP($\mathcal{X}$) carry over to CLP(Σ):

Theorem (Soundness of CLP(Σ))

The calculus for CLP(Σ) is sound

Completeness of CLP(Σ)

Completeness means the following

Theorem (Completeness of CLP(Σ))

If $\Pi \cup \Sigma \models (\forall)\mathcal{S} \supset \Gamma$ *and* $\mathcal{S}$ *is satisfiable wrt.* Σ*, then there are successful derivations for* Γ *with computed answer constraint stores* $\mathcal{S}_1, \ldots, \mathcal{S}_n$ *such that* $\Sigma \models (\forall)(\mathcal{S} \supset (\mathcal{S}_1 \vee \ldots \vee \mathcal{S}_n))$*.*

- The answers are conditional on the computed constraint store (not necessary)
- Maybe we need multiple derivations

Sample Derivations

Let $\Sigma = \{\exists x P(x)\}$.

The ALP query `?(p(X))` ends with constraint store `?(p(X))`.

Consider the ALP Π

```
p :- ?(a).
p :- ?(-a).
```

Of course $\Pi \models P$. But you need two derivations to show this:

$$A \vee \neg A \models P$$

Using ALPs

Planning with ALPs

Planning problems are easily formulated:

```
plan :- ?(goal).
plan :- do(A), plan.
```

Answer the query `?- plan.`? Prove plan existence

Disjunctive substitutions: Conditional planning

Online vs Offline Reasoning

Planning is Offline Reasoning: Proof Search

Online Agent Control: Attempt proof, try not to get stuck, do not use disjunctive substitutions

Own Implementation Work

We use mature Prolog technology for resolution

We implemented two action reasoners:

(1) Use DL action calculi

(2) Use FO over finite domain (propositional logic)

Agent Logic Programs ...

- ... specify strategies for offline planning and online agent control
- ... have logic based world model
- ... are largely independent from particular action calculi
- ... have nice declarative semantics

Thanks for your attention!