Ph.D Dissertation Oral Defense

Representing Actions in Logic-based Languages

Fangkai Yang
Supervisor: Vladimir Lifschitz
Department of Computer Science
The University of Texas at Austin

Mar, 26, 2014
Knowledge about Actions

• This dissertation is about representing knowledge about dynamic domains.

• In dynamic domains, we distinguish between
  – **fluents**: anything that depends on the state of the world and can be affected by performing actions.
  – **actions**: anything that can be executed and may change the state of the world

• Dynamic domains can be described in logic programs under stable model semantics (Gelfond & Lifschitz, 1988), causal theories (McCain & Turner, 1997), and action languages $B$ (Gelfond & Lifschitz, 1998) and $C+$ (Giunchiglia et al, 2004).
Overview of the Thesis

• We study mathematical properties of causal theories related to synonymy and non-definite causal rules (Chapters 3–5).

• We use the stable model semantics to design and study a new action language $BC$ that combines the most useful expressive possibilities of languages $B$ and $C$ (Chapters 6–9).

• We use $BC$ to reimplement the USA/RCS-Advisor, and to implement a task planner for mobile robots (Chapters 10–12).
Plan of the Talk

- Review: semantics of logic programs.
- $\Gamma$-tight logic programs.
- Action language $BC$.
- Describing the RCS system in $BC$.
- Task planner for mobile robots.
Plan of the Talk

- Review: semantics of logic programs.
- $\Gamma$-tight logic programs.
- Action language $BC$.
- Describing the RCS system in $BC$.
- Task planner for mobile robots.
An Overview

- Early work on semantics of logic programming aimed at characterizing the behavior of Prolog system.

- The completion semantics (Clark, 1978) and stable model semantics (Gelfond & Lifschitz, 1988) are two such attempts.

- Answer set solvers are software systems that generate stable models.

- Finally, a new programming paradigm: answer set programming was born (Niemela, 1999; Marek & Truszczynski, 1999).
Consider the following Prolog program

\[ \text{p(a). q(b).} \]
\[ \text{p(X) :- q(X).} \]

- Prolog will answer true to the queries \( \text{p(a), p(b), q(b)} \), and false to \( \text{q(a)} \).

- Answer set solver CLINGO will return the following stable model:

  Answer: 1
  \[ \text{p(a) q(b) p(b)} \]
CLINGO can also handle the program

\[ \text{p(a). q(b).} \]
\[ \{ \text{p(X)} \} :- \text{q(X).} \text{ % choice rule} \]

by returning two stable models:

Answer: 1
p(a) q(b)
Answer: 2
p(a) q(b) p(b)

But choice rules are not supported by Prolog.
For program

\[ p:~ \text{not} \ q. \]
\[ q:~ \text{not} \ p. \]

CLINGO returns two stable models:

Answer: 1
\[ p \]
Answer: 2
\[ q \]

Prolog system replies: Error: local stack overflow.
Two Approaches to the Semantics of Logic Programs

- (Clark, 1978) proposed the notion of program completion to characterize logic programs in terms of first-order theories.
- (Lloyd & Topor, 1984) extend the program completion to a wider class of logic programs.
- (Gelfond & Lifschitz, 1988) proposed the stable model semantics.
- (Ferraris et al, 2007, 2011) extended the stable model semantics to a wider class of logic programs.
A Lloyd-Topor program is a finite set of rules of the form

\[ p(t) \leftarrow G \]

where \( G \) is a first-order formula.

Example:

\[
\begin{align*}
p(a), \\
q(b), \\
p(x) &\leftarrow q(x)
\end{align*}
\]

is a Lloyd-Topor program. Choice rules such as

\[
\{p(x)\} \leftarrow q(x)
\]

are not allowed in Lloyd-Topor programs.
Let $\Pi$ be a Lloyd-Topor program, and $p$ a predicate constant. Let

$$p(t^i) \leftarrow G^i \quad (i = 1, 2, \ldots)$$

be all rules of $\Pi$ that contain $p$ in the head. The completed definition of $p$ is the formula

$$\forall x \left( p(x) \leftrightarrow \bigvee_i \exists y^i(x = t^i \land G^i) \right)$$

where $x$ is a list of distinct variables not appearing in any of the rules (1), and $y^i$ is the list of free variables in (1).

The completion of $\Pi$, denoted by $\text{Comp}[\Pi]$, is the conjunction of the completed definitions of all predicate constants.
For program $\Pi_1$,

\[ p(a), \]
\[ q(b), \]
\[ p(x) \leftarrow q(x) \]

its program completion $\text{Comp}[\Pi_1]$ is

\[ \forall x_1(p(x_1) \leftrightarrow x_1 = a \lor \exists x(x_1 = x \land q(x))), \]
\[ \forall x_1(q(x_1) \leftrightarrow x_1 = b). \]

It is equivalent to

\[ \forall x(p(x) \leftrightarrow x = a \lor q(x)), \]
\[ \forall x(q(x) \leftrightarrow x = b). \]
First-order Stable Models

• (Ferraris et al., 2011) defined stable models by a stable model operator $SM$.  
• A first-order sentence is a rule if it has the form $\forall(F \rightarrow G)$ and has no occurrence of $\rightarrow$ other than the one explicitly shown.
• A logic program is a conjunction of rules.
• For instance, program

$$p(a),$$
$$q(b),$$
$$p(x) \leftarrow q(x)$$

is identified with the sentence

$$p(a) \land q(b) \land \forall x(q(x) \rightarrow p(x)).$$
First-order Stable Models, continued

- Let \( p \) be a list of predicate constants; members of \( p \) will be called \textit{intensional predicates}. For any logic program \( F \), \( SM_p[F] \) is a conjunction of \( F \) with a certain second-order sentence.

- A model of \( F \) is \textit{stable} relative to \( p \) if it satisfies \( SM_p[F] \).

- For instance, \( SM_{pq}[\Pi_1] \) is

\[
p(a) \land q(b) \land \forall x (q(x) \rightarrow p(x)) \land \text{some second-order sentence.}
\]

It turns out that this formula is equivalent to \( \text{Comp}[\Pi_1] \): 

\[
\forall x (p(x) \leftrightarrow x = a \lor q(x)),
\]

\[
\forall x (q(x) \leftrightarrow x = b).
\]
Another Example

Program $\Pi_2$

\[
p(a), \\
q(b), \\
\{p(x)\} \leftarrow q(x)
\]

is identified with the sentence

\[
p(a) \land q(b) \land \forall x(q(x) \rightarrow p(x) \lor \neg p(x)).
\]

$SM_{pq}[\Pi_2]$ is equivalent to the first-order sentence

\[
p(a) \land \forall x(p(x) \rightarrow x = a \lor x = b) \land \forall x(q(x) \leftrightarrow x = b).
\]
Plan of the Talk

• Review: semantics of logic programs.

• $\Gamma$-tight logic programs.

• Action language $BC$.

• Describing the RCS system in $BC$.

• Task planner for mobile robots.
• (Fages, 1994) defined the concept of a tight program and proved that for tight programs, the stable model semantics is equivalent to completion semantics. His result was generalized in (Ferraris et al, 2011; Lee & Meng, 2011).

• These theorems sometimes allow us to transform SM[II] to an equivalent first-order sentence.

• The theory of Γ-tight Lloyd-Topor programs developed in Chapters 8 and 9 of the dissertation further extends this line of the work.
• **Tight programs** defined in earlier work of Fages’ theorem refer to predicate dependency graphs. For example, the predicate dependency graph of program

\[
p(a, b),
q(x, y) \leftarrow p(y, x) \land \neg p(x, y)
\]

is

\[\begin{array}{c}
q \\
\rightarrow \\
p
\end{array}\]

• Our definition of Γ-tight programs refers to rule dependency graphs:

\[
q(x_1, y_1) \leftarrow p(y_1, x_1) \land \neg p(x_1, y_1)
\]

\[
\downarrow p(y_1, x_1)
\]

\[p(a, b)\]

• Theorem 8.2.1 is a version of Fages’ theorem that refers to rule dependency graphs. Its proof uses the theory of infinitary programs due to Truszczynski.
Example: Moving Objects

We consider a logic program $M$ that describes moving objects over the time instants $0,...,k$.

(i) the facts

$$\text{step}(0), \text{step}(1), \ldots, \text{step}(k);$$
$$\text{next}(0,1), \text{next}(1,2), \ldots, \text{next}(k-1,k);$$

(ii) the unique name constraints

$$\leftarrow \hat{i} = \hat{j} \quad (1 \leq i < j \leq k);$$

(iii) the constraints describing the arguments of $at$ and $move$:

$$\leftarrow at(x,y,z) \land \neg(object(x) \land place(y) \land step(z))$$

and

$$\leftarrow move(x,y,z) \land \neg(object(x) \land place(y) \land step(z));$$
(iv) the existence of location constraint

\[ \leftarrow \text{object}(x) \land \text{step}(z) \land \neg \exists y \ \text{at}(x, y, z); \]

(v) the uniqueness of location constraint

\[ \leftarrow \text{at}(x, y_1, z) \land \text{at}(x, y_2, z) \land y_1 \neq y_2; \]

(vi) the rule expressing the effect of moving an object:

\[ \text{at}(x, y, u) \leftarrow \text{move}(x, y, z) \land \text{next}(z, u); \]

(vii) the choice rule expressing that initially an object can be anywhere:

\[ \{ \text{at}(x, y, 0) \} \leftarrow \text{object}(x) \land \text{place}(y); \]

(viii) the choice rule expressing by default the location of an object does not change.

\[ \{ \text{at}(x, y, u) \} \leftarrow \text{at}(x, y, z) \land \text{next}(z, u). \]
Characterizing the Stable Models of $M$

$\Gamma$: the conjunction of the constraints from $M$.

**Proposition** $SM_p[M]$ is equivalent to the conjunction of $\Gamma$ with the universal closures of the formulas

$$\text{step}(z) \leftrightarrow \bigvee_{i=0}^{k} z = \hat{i},$$

$$\text{next}(z, u) \leftrightarrow \bigvee_{i=0}^{k-1} (z = \hat{i} \land u = \hat{i+1}),$$

$$\text{at}(x, y, \hat{i+1}) \leftrightarrow (\text{move}(x, y, \hat{i}) \lor (\text{at}(x, y, \hat{i}) \land \neg \exists w \text{ move}(x, w, \hat{i}))) \quad (i = 0, \ldots, k - 1).$$

The last equivalence is similar to successor state axioms in the sense of Reiter. The proof involves transforming $M$ into an equivalent $\Gamma$-tight Lloyd-Topor program, and simplifying the first-order formula given by our extension of Fages’ theorem.
Plan of the Talk

- Review: semantics of logic programs.
- $\Gamma$-tight logic programs.
- Action language $\mathcal{BC}$.
- Describing the RCS system in $\mathcal{BC}$.
- Task planner for mobile robots.
Action Languages

- Action languages have a more concise syntax than logic programs.
- Example: in an action language, the rule

\[ at(x, y, u) \leftarrow move(x, y, z) \land next(z, u) \]

will be written as

\( move(x, y) \text{ causes } at(x, y). \)

The rule

\[ \{ at(x, y, u) \} \leftarrow at(x, y, z) \land next(z, u) \]

will be written as

\( \text{inertial } at(x, y). \)
Any action description describes a transition system: a directed graph whose vertices correspond to states, and edges correspond to the execution of actions.

A path in a transition system represents the execution of a plan.
Action Language $BC$ - Syntax

- An action description in the language $BC$ includes a finite set of symbols of two kinds:
  - fluent constants: regular fluents and statically determined fluents.
  - action constants.

- A finite set of cardinality $\geq 2$, called the domain, is assigned to every fluent constant.

- An atom is an expression of the form $f = v$, where $f$ is a fluent constant, and $v$ is an element of its domain. If the domain is \{false, true\}, then $f$ is called Boolean.

- For a Boolean fluent $f$, $f = true$ can be abbreviated as $f$, and $f = false$ can be abbreviated as $\sim f$. 

A static law is an expression of the form

\[
\underbrace{A_0}_{\text{conclusion}} \text{ if } \underbrace{A_1, \ldots, A_m}_{\text{premises}} \text{ ifcons } \underbrace{A_{m+1}, \ldots, A_n}_{\text{justifications}}
\]

\((n \geq m \geq 0)\), where each \(A_i\) is an atom.

A dynamic law is an expression of the form

\[
A_0 \text{ after } A_1, \ldots, A_m \text{ ifcons } A_{m+1}, \ldots, A_n
\]

\((n \geq m \geq 0)\), where

- \(A_0\) is an atom containing a regular fluent constant,
- each of \(A_1, \ldots, A_m\) is an atom or an action constant, and
- \(A_{m+1}, \ldots, A_n\) are atoms.

An action description in the language of \(\mathcal{BC}\) is a finite set consisting of static and dynamic laws.
Abbreviations

Examples: \textit{move}(x, y) \textbf{causes} \textit{at}(x, y) \textbf{stands for the dynamic law}

\[ \textit{at}(x, y) \textbf{ if } \textit{move}(x, y) \textbf{ if cons } \top; \]

\textbf{inertial} \textit{at}(x, y) \textbf{stands for the pair of dynamic laws}

\[ \textit{at}(x, y) = \text{true} \textbf{ if } \textit{at}(x, y) = \text{true} \textbf{ if cons } \textit{at}(x, y) = \text{true}, \]
\[ \textit{at}(x, y) = \text{false} \textbf{ if } \textit{at}(x, y) = \text{false} \textbf{ if cons } \textit{at}(x, y) = \text{false}. \]

or more concisely,

\[ \textit{at}(x, y) \textbf{ if } \textit{at}(x, y) \textbf{ if cons } \textit{at}(x, y), \]
\[ \sim \textit{at}(x, y) \textbf{ if } \sim \textit{at}(x, y) \textbf{ if cons } \sim \textit{at}(x, y). \]
Example: The Blocks World

- Let $Blocks$ be a finite non-empty set of symbols.
- The action description uses
  - for each $B \in Blocks$, regular fluent constant $Loc(B)$ with domain $Blocks \cup \{Table\}$, and statically determined Boolean fluent constant $InTower(B)$;
  - for each $B \in Blocks$ and each $L \in Blocks \cup \{Table\}$, action constant $Move(B, L)$. 
Static Laws

• Two different blocks cannot rest on the same block:

\[ \text{impossible } \text{Loc}(B_1) = B, \text{Loc}(B_2) = B \quad (B_1 \neq B_2). \]

• The definition of \( InTower(B) \):

\[
\begin{align*}
\text{InTower}(B) & \quad \text{if } \text{Loc}(B) = \text{Table}, \\
\text{InTower}(B) & \quad \text{if } \text{Loc}(B) = B_1, \text{InTower}(B_1), \\
\text{default} & \quad \sim \text{InTower}(B).
\end{align*}
\]

• Blocks don’t float in the air:

\[ \text{impossible } \sim \text{InTower}(B). \]
Dynamic Laws

- The commonsense law of inertia:

  \textbf{inertial} \ Loc(B).

- The effect of moving a block:

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

- A block cannot be moved if there are other blocks above it:

  \textbf{nonexecutable} \ Move(B, L) \textbf{ if} \ Loc(B_1) = B.
Example: Non-inertial Domain of Leaking Container

- We consider a container with capacity up to $n$ that has a leak, so that it loses $k$ units of liquid per unit of time, unless more liquid is added.

- We use regular fluent constants $Amt$ with domain $\{0, \ldots, n\}$, for the amount of liquid in the container, and the action constant $FillUp$.

- Dynamic laws:

  \[
  \text{default } Amt = \max(a - k, 0) \text{ after } Amt = a \ (a = 0, \ldots, n),
  \]

  $FillUp$ causes $Amt = n$. 

Representing Actions in Logic-based Languages Mar, 26, 2014
Language $\mathcal{BC}$: Semantics

- For every action description $D$, we will define a sequence of logic programs $PN_0(D), PN_1(D), \ldots$.
  - The stable models of $PN_0(D)$ represent the states.
  - The stable models of $PN_1(D)$ represent the transitions.
  - More generally, the stable models of $PN_l(D)$ represent paths of length $l$ in the transition system.

- The vocabulary of $PN_l(D)$ consists of
  - expressions $i:A$ for nonnegative integers $i \leq l$ and all atoms $A$, and
  - expressions $i:a$ for nonnegative integers $i < l$ and all action constants $a$. 
Language $\mathcal{BC}$: Semantics

The semantics of a $\mathcal{BC}$-description is defined by the stable models of the logic program consisting of

- rules

\[
i : A_0 \leftarrow i : A_1 \land \cdots \land i : A_m \land \neg \neg i : A_{m+1} \land \cdots \land \neg \neg i : A_n, \quad (i \leq l)
\]

for all static laws

\[
A_0 \ \textbf{if} \ A_1, \ldots, A_m \ \textbf{ifcons} \ A_{m+1}, \ldots, A_n,
\]

- rules

\[
i + 1 : A_0 \leftarrow i : A_1 \land \cdots \land i : A_m \land \neg \neg (i + 1) : A_{m+1} \land \cdots \land \neg \neg (i + 1) : A_n \quad (i < l)
\]

for all dynamic laws

\[
A_0 \ \textbf{after} \ A_1, \ldots, A_m \ \textbf{ifcons} \ A_{m+1}, \ldots, A_n,
\]
Language $\mathcal{BC}$: Semantics, cont.

- the choice rule $\{0 : A\}$ for every atom $A$ containing a regular fluent constant,
- the choice rule $\{i : a\}$ for every action constant $a$ and every $i < l$,
- the existence of value constraint

\[ \leftarrow \neg i : (f = v_1) \land \cdots \land \neg i : (f = v_k) \]

for every fluent constant $f$ and every $i \leq l$, where $v_1, \ldots, v_k$ are all elements of the domain of $f$,
- the uniqueness of value constraint

\[ \leftarrow i : (f = v) \land i : (f = w) \]

for every fluent constant $f$, every pair of distinct elements $v, w$ of its domain, and every $i \leq l$. 
Language BC: Semantics, cont.

Given a BC-description $D$, the transition system $T(D)$ is defined as follow:

- For every stable model $X$ of $PN_0(D)$, the set of atoms $A$ such that $0:A \in X$ is a state of $T(D)$

- For every stable model $X$ of $PN_1(D)$, $T(D)$ includes the transition $\langle s_0, \alpha, s_1 \rangle$, where $s_i$ ($i = 0, 1$) is the set of atoms $A$ such that $i:A$ belongs to $X$, and $\alpha$ is the set of action constants $a$ such that $0:a$ belongs to $X$.

**Theorem 6.3.1** For every transition $\langle s_0, \alpha, s_1 \rangle$, $s_0$ and $s_1$ are states.

Theorem 6.3.2 expresses that the stable models of $PN_l(D)$ are in a 1-1 correspondence with the paths of length $l$ in the transition system.
The Leaking Container Domain in the Language of CLINGO

% declarations of variables for steps and amounts
step(0..1). #domain step(I).
amount(0..n). #domain amount(A).

% translations of dynamic laws
{amt(AA,I+1)} :- amt(A,I), AA=(|A-k|+(A-k))/2, I<l.
amt(n,I+1) :- fillup(I), I<l.

% standard choice rules
{amt(A,0)}.
{fillup(I)} :- I<l.

% existence and uniqueness of value constraints
:- {amt(AA,I) : amount(AA)} 0.
:- 2 {amt(AA,I) : amount(AA)}. 
Temporal Projection

At time 0, a container of capacity 10 is full. It loses 3 units of liquid per unit of time. At time 3, it is filled up. Describe how the amount of liquid in the container will change with time.

\[
\text{amt}(10,0).
\]
\[
\text{fillup}(3). \quad -\text{fillup}(0..2;4..1).
\]

The output of the solver \texttt{CLINGO} includes the following atoms:

\[
\text{amt}(10,0) \quad \text{amt}(7,1) \quad \text{amt}(4,2) \quad \text{amt}(1,3) \quad \text{amt}(10,4) \\
\text{amt}(7,5) \quad \text{amt}(4,6) \quad \text{amt}(1,7) \quad \text{amt}(0,8) \quad \text{amt}(0,9)
\]
Relation to Action Languages $\mathcal{B}$ and $\mathcal{C}^+$

- Theorem 6.8.1 shows that $\mathcal{B}$ is a subset of $\mathcal{BC}$.
- Theorem 6.9.1 describes a common subset of $\mathcal{BC}$ and $\mathcal{C}^+$.
- $\mathcal{BC}$ extends $\mathcal{B}$ with non-inertial and multi-valued fluents.
- $\mathcal{BC}$ expresses recursive definitions more easily than $\mathcal{C}^+$:

  \[ \text{InTower}(B) \text{ if } \text{Loc}(B) = \text{Table}, \]
  \[ \text{InTower}(B) \text{ if } \text{Loc}(B) = B_1, \text{InTower}(B_1). \]
Plan of the Talk

- Review: semantics of logic programs.
- Γ-tight logic programs.
- Action language $\mathcal{BC}$.
- Describing the RCS system in $\mathcal{BC}$.
- Task planner for mobile robots.
Input Language of **CPLUS2ASP**

- Software system **CPLUS2ASP** was created at ASU to answer queries about action descriptions in $C+$. Its Version 2 is applicable also to the language $BC$.

- Chapter 10 describes the precise relationship between language $BC$ and a subset of the input language of **CPLUS2ASP**.
A Sample CPLUS2ASP Input

:- sorts
cloc >> cblock.

:- constants
cloc(cblock) :: cinertialFluent(cloc);
cmove(cblock, cloc):: caction.

:- variables
CB :: cblock;
CL :: cloc.

% Moving a block
cmove(CB, CL) causes clcloc(B)=CL.
Reactive Control System of the Space Shuttle

- The reactive control system of the space shuttle is used to control the space shuttle to perform various maneuvers.
- It contains a hydraulic system with tanks, valves, switches that are connected to jets.
- Astronauts send commands through valve control system to open or close valves so that fuel and oxidizers are mixed, certain jets are fired, and maneuvers are achieved.
- USA/RCS-Advisor system (joint project of Texas Tech University and NASA, 2000-2006) was used to compute such list of commands for a given maneuvering mission. It can be used both when all devices work correctly and one some of them are faulty.
Describing the USA/RCS-Advisor in $\mathcal{BC}$

- In USA/RCS-Advisor, the dynamic domain was formalized using ASP. It involves recursively defined fluents and statically determined fluents, and it could not be expressed in an action language at that time.

- We reimplemented the USA/RCS-Advisor in the subset of \texttt{cplus2asp} that corresponds to language $\mathcal{BC}$. This illustrates the expressivity of $\mathcal{BC}$.

- We used \texttt{cplus2asp} and \texttt{clingo} to generate plans in this domain and tested the correctness of our formalization by comparing the stable models produced by both implementations.
Comparison with ASP Encoding

- A multi-valued inertial fluent \texttt{in\_state(Dev)} describes the state of the device \texttt{Dev}.

- In the \texttt{CPLUS2ASP} encoding, it is declared as follows:
  \begin{verbatim}
in\_state(device):: inertialFluent(state);
\end{verbatim}

- In the ASP encoding, three rules correspond to this declaration:
  \begin{verbatim}
h(in\_state(D,S),T1) :- next(T,T1), of\_type(D,Dev), state\_of(S,Dev),
  h(in\_state(D,S),T), not -in\_state(D,S,T1).

nh(in\_state(D,S),T) :- time(T), of\_type(D,Dev), state\_of(S,Dev),
  state\_of(S1,Dev), neq(S,S1), h(in\_state(D,S1),T).
  :- time(T), of\_type(D,Dev), state\_of(S,Dev),
  h(in\_state(D,S),T), nh(in\_state(D,S),T).
\end{verbatim}
Comparison of Running Times

The following table compares \texttt{cplus2asp} encoding with the original ASP encoding:

<table>
<thead>
<tr>
<th></th>
<th>\texttt{cplus2asp} encoding</th>
<th>ASP encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>grounding time (sec)</td>
<td>28.179</td>
<td>0.056</td>
</tr>
<tr>
<td>solving time (sec)</td>
<td>5.281</td>
<td>0.03</td>
</tr>
<tr>
<td>ground file size</td>
<td>118M</td>
<td>140K</td>
</tr>
</tbody>
</table>

This comparison shows that \texttt{cplus2asp} needs significant improvement before it can be applied to large domains.
Plan of the Talk

- Review: semantics of logic programs.
- $\Gamma$-tight logic programs
- Action language $BC$.
- Describing the RCS system in $BC$.
- Task planner for mobile robots.
The robot needs to visit people to collect their mail, and people can pass mail to others.

Information may be incomplete. Planning for human-robot interaction is needed.

Each action has a cost (its execution time). Lowest cost plans are needed.
Architecture

Answer Set Solver (CLINGO)

Planner

Actions

Executor

Observations

Sensor Data

Environment

Actions
Rigid Knowledge

- **hasdoor**\((R, D)\): room \(R\) has door \(D\).

\[
\begin{align*}
\text{hasdoor}(o_1, d_1) & \quad \text{hasdoor}(o_2, d_2) & \quad \text{hasdoor}(o_3, d_3) \\
\text{hasdoor}(\text{lab}_1, d_4) & \quad \text{hasdoor}(\text{lab}_1, d_5)
\end{align*}
\]

**default** \(\sim \text{hasdoor}(R, D)\)

- **acc**\((R_1, D, R_2)\): room \(R_1\) is accessible from room \(R_2\) through door \(D\).

\[
\begin{align*}
\text{acc}(R, D, \text{cor}) & \quad \text{if} \quad \text{hasdoor}(R, D) \\
\text{acc}(R, D, \text{cor}) & \quad \text{if} \quad \text{acc}(\text{cor}, D, R) \\
\text{default} & \quad \sim \text{acc}(R_1, D, R_2)
\end{align*}
\]

- **knows**\((P_1, P_2)\): \(P_1\) knows where person \(P_2\) is.

- **passto**\((P_1, P_2)\): person \(P_1\) has passed outgoing mail to person \(P_2\).
Time-dependent Knowledge

- \(\text{inside}(P, R):\) person \(P\) is inside room \(R\).

\[
\sim \text{inside}(P, R_2) \text{ if } \text{inside}(P, R_1) \quad (R_1 \neq R_2)
\]

**inertial** \(\text{inside}(P, R)\)

- \(\text{knowinside}(P, R):\) the robot knows that person \(P\) is inside room \(R\).

\[
\sim \text{knowinside}(P, R_2) \text{ if } \text{knowinside}(P, R_1) \quad (R_1 \neq R_2)
\]

**inertial** \(\text{knowinside}(P, R)\).

\(\text{inside}(P, R) \text{ if } \text{knowinside}(P, R)\)

- \(\text{mailcollected}(P):\) person \(P\)'s mail has been collected.

\[
\text{mailcollected}(P_1) \text{ if } \text{mailcollected}(P_2), \text{passto}(P_1, P_2)
\]

- Other fluents: \(\text{facing}(D), \text{beside}(D), \text{loc} = R, \text{open}(D), \text{visiting}(P)\).
Action Knowledge

- $\text{approach}(D)$: the robot approaches door $D$.

  $\text{approach}(D) \text{ causes } \text{facing}(D)$

  nonexecutable $\text{approach}(D)$ if $\text{loc} = R, \sim \text{hasdoor}(R, D)$

  nonexecutable $\text{approach}(D)$ if $\text{facing}(D)$

- $\text{collectmail}(P)$: the robot collects mail from person $P$.

  $\text{collectmail}(P) \text{ causes } \text{mailcollected}(P)$

- $\text{askploc}(P_1, P)$: the robot asks $P_1$ about the location of $P$.

  $\text{askploc}(P_1, P) \text{ causes } \text{knowinside}(P, R) \text{ if } \text{inside}(P, R)$

- Other actions: $\text{gothrough}(D)$, $\text{greet}(P)$, $\text{opendoor}(D)$. 
Planning

- Initial values of some fluents (*knowinside*, *knows*, and *passto*) come from tables that are maintained by the planner.

- Initial values of other fluents (*loc*, *beside*, *facing*) are obtained from sensors.

- The goal condition is given by the user.

- **CLINGO** is called repeatedly to look for plans of increasing length until a plan is found.

- The plan is sent to the executor. Execution is monitored. If a failure occurs, a new plan will be generated.
Example 1: Visiting Alice

- The goal condition is
  \[ \text{maxLength} : \text{visiting}(\text{Alice}) \]

- When \( \text{maxLength} = 7 \), the following plan is generated by CLINGO:

  \[
  0 : \text{approach}(d_5),\ 1 : \text{opendoor}(d_5),\ 2 : \text{gothrough}(d_5),\ \\
  3 : \text{approach}(d_1),\ 4 : \text{opendoor}(d_1),\ 5 : \text{gothrough}(d_1),\ \\
  6 : \text{greet}(\text{alice})
  \]
Example 2: Mail Collection

- Given \textit{passto} relation: \textit{passto}(bob, alice), \textit{passto}(dan, bob). Dan’s location is not known.

- The goal is to collect mail from everyone.

- The following plan is generated:

  0: \textit{approach}(d_5), 1: \textit{opendoor}(d_5), 2: \textit{gothrough}(d_5),

  3: \textit{approach}(d_1), 4: \textit{opendoor}(d_1), 5: \textit{gothrough}(d_1), 6: \textit{collectmail}(alice),

  7: \textit{approach}(d_1), 8: \textit{opendoor}(d_1), 9: \textit{gothrough}(d_1),

Example 3: Mail Collection with Human-Robot Interaction

• But what if Dan, whose location is unknown, does not pass his mail to anyone?

• As part of the domain knowledge, the robot has the $knows$ relation:
  \[ knows(carol, dan) \]

• The following plan is generated:

  0: approach($d_5$), 1: opendoor($d_5$), 2: gothrough($d_5$), 3: approach($d_1$),
  4: opendoor($d_1$), 5: gothrough($d_1$), 6: collectmail($alice$), 7: approach($d_1$),
  8: opendoor($d_1$), 9: gothrough($d_1$), 10: approach($d_3$), 11: opendoor($d_3$),
  12: gothrough($d_3$), 13: collectmail($carol$), 14: greet($carol$),
  15: askploc($carol, dan$), 16: collectmail($dan$)
Generating Lowest Cost Plan

- When the robot navigates through a building, it is possible that the plan of minimal length does not lead to the shortest navigational distance.
- The cost of an action is the estimated execution time. Our goal is to find the lowest cost plan.
- In the example domain, the costs of all actions except for approach are fixed. The cost of approach($D$) when the robot is currently in room $R$ and beside door $D_1$ is expressed by an external function @cost($D_1$, $R$, $D$).
- During search, CLINGO will query an external procedure to obtain the value of the function. In our architecture, the value is provided by the learning module.
- The lowest cost plan is generated using CLINGO’s optimization statements.
Building-wide Intelligence

- The framework is implemented and tested both in simulation and real domains. It is part of the Building-wide Intelligence project led by Peter Stone.

Figure 1: BWI robot

Figure 2: 3NE Wing of GDC
Summary

• Theoretical aspects:
  – Rule dependency graphs can be used to describe stable models by first-order formulas.
  – This method is applicable to logic programs representing dynamic domains.
  – Action language $BC$ improves on the earlier proposals $B$ and $C+$.

• Practical aspects:
  – $BC$ is used to reimplement a fragment of the USA/RCS-Advisor.
  – $BC$ is used to implement a task planner for mobile robots. The planner integrates planning, execution monitoring, replanning and human-robot interaction, and can generate lowest-cost plans.
Other Parts of the Thesis

• Chapter 3: Functional completion.
• Chapter 4: Representing first-order causal theories in logic programming.
• Chapter 5: Eliminating functions from a causal theory.
• Chapter 9: Propositional infinitary logic programs.


