Modular Action Language $\mathcal{ALM}$

Michael Gelfond and Daniela Inclezan
Computer Science Department
Texas Tech University

February 11, 2010
Specifying Dynamic Systems

• This talk discusses the problem of representing knowledge about discrete dynamic systems modeled by transition diagrams.

\[ \langle s_0, a, s_1 \rangle \in T \]

iff the execution of action \( a \) in a state \( s_0 \) may move the system to state \( s_1 \).

• Action languages are used for specifying such diagrams.
Action Languages and ASP

We view representing dynamic systems in action languages as part of ASP:

- Action theory $A$ describing dynamic system $S$ can be viewed as a front-end for an ASP program $\Pi$ describing the same system.
- $\Pi$ plays the major role in many applications of ASP including planning and diagnostics.
Action Languages $\mathcal{AL}$ and $\mathcal{C}$

The semantics of $\mathcal{AL}$ incorporates the *inertia* axiom – “Things normally stay the same” and is very close to ASP.

$\mathcal{C}$ incorporates the *causality principle* – “Everything true in the world must be caused” and is based on causal logic.

The addition of a state constraint

$$f \text{ if } f$$

does not change a theory of $\mathcal{AL}$ but does change a theory of $\mathcal{C}$. 
Need for Modules

Both languages lack the structure needed for expressing the hierarchies of abstractions often necessary for the design of larger knowledge bases.

The addition of such a structure will facilitate the reuse of knowledge and the organization of libraries.
Example

• Often actions are defined in terms of other actions, e.g.

move – “change position”

carry – “move while holding”

push – “carry by applying effort”.

• We should have a convenient way for representing this hierarchy.

• \textit{MAD} – an extension of \textit{C} aimed for this purpose; \textit{ALM} – does the same for \textit{AL}.
Modules of $\mathcal{ALM}$

Syntactically a module can be viewed as a collection of declarations of sort, fluent and action classes of the system.

```
module name
    sort declarations
    fluent declarations
    action declarations
```
Example: move_between_areas

module move_between_areas

sort declarations

  things : sort
  movers : things
  areas : sort

fluent declarations

  loc_in(things, areas) : inertial fluent

axioms

  \neg loc_in(T, A_2) \text{ if } disjoint(A_1, A_2),
  
  loc_in(T, A_1).

end of loc_in
action declarations

move : action

attributes

actor : movers

origin, dest : areas

axioms

move causes loc_in(O, A) if actor = O,
    dest = A.

impossible move if actor = O,
    origin = A,
    \neg loc_in(O, A).

impossible move if origin = A_1,
    dest = A_2,
    \neg disjoint(A_1, A_2).

end of move
Comments

• The actions of a module are action \textit{classes}.

• The sorts, fluents, actions, and axioms of the module are \textit{uninterpreted}.

• Semantically, a collection of modules can be viewed as a \textit{mapping} of possible interpretations of the symbols of the domain into the transition diagram describing a dynamic system.

• A \textit{system description} is a set of modules followed by an interpretation of its symbols.

• Modules can be combined into \textit{libraries} and imported from there using \textit{import} statements.
Interpreting the Symbols

structure of basic_travel

sorts
michael, bob in movers
london, paris, rome in areas

actions

instance move(O, A₁, A₂) : move
actor := O
origin := A₁
dest := A₂

statics

disjoint(london, paris).
disjoint(paris, rome).
disjoint(rome, london).
Actions as Special Cases

module carrying_things

sort declarations
  areas, things : sort
  movers, carriables : things

fluent declarations

  holding(things, things) : inertial fluent
  loc_in(things, areas) : inertial fluent

axioms

  loc_in(T, A) ≡ loc_in(O, A) if holding(O, T).

end of loc_in
action declarations

\textit{carry} : move

attributes

\textit{carried\_thing} : \textit{carriables}

axioms

impossible \textit{carry} \textbf{if} \hspace{1em} actor = O, \hspace{1em} carried\_thing = T, \hspace{1em} \neg \text{holding}(O, T).

end of \textit{carry}
More on Formal Semantics of $\mathcal{ALM}$

A system description $S$ of $\mathcal{ALM}$ is mapped into ground statements of the non-modular language $\mathcal{AL}$ which uniquely define the transition diagram of $S$. For example, the action instance $move(bob, london, paris)$ and the $\mathcal{ALM}$ causal law

\[
move \text{ causes } loc\_in(O, A) \text{ if } actor = O, \quad dest = A.
\]

are turned into the $\mathcal{AL}$ causal law

\[
move(bob, london, paris) \text{ causes } loc\_in(bob, paris)
\]
Describing a System’s History

A system description $S$ of $\mathcal{ALM}$ is normally used in conjunction with the description of the system’s history, $\mathcal{H}$ – a collection of facts of the form:

$happened(a, i)$
$observed(f, true/false, i)$
$intend([a_1, \ldots, a_n], i)$

Together $S$ and $\mathcal{H}$ define the collection of possible trajectories of the system up to the current step $n$.

These trajectories can be extracted from answer sets of the translation $\Pi$ of $S$ together with axioms for $happened$, $observed$, and $intend$. 
Axioms for Intentions

Chitta Baral and Michael Gelfond.
Reasoning about intended actions.
In AAAI’05, pages 689-694, 2005.

Michael Gelfond.
Going places - notes on a modular development of knowledge about travel.
Axioms for Intentions

1. Normally intended actions are executed the moment that such an execution becomes possible.

\[ \text{occurs}(A, I) \leftarrow \text{intend}(A, I), \]
\[ \not \neg \text{occurs}(A, I). \]

2. Unfulfilled intentions persist:

\[ \text{intend}(A, I + 1) \leftarrow \text{intend}(A, I), \]
\[ \neg \text{occurs}(A, I), \]
\[ \not \neg \text{intend}(A, I + 1). \]
Axioms for Intentions

Axioms

\[ \text{intend}(V, I) \leftarrow \text{intend}(S, I), \]
\[ \text{component}(V, 1, S). \]

\[ \text{intend}(V_2, I_2) \leftarrow \text{intend}(S, I_1), \]
\[ \text{component}(V_2, K + 1, S), \]
\[ \text{component}(V_1, K, S), \]
\[ \text{ends}(V_1, I_2). \]

\[ \text{ends}(S, I) \leftarrow \text{length}(S, N), \]
\[ \text{component}(V, N, S), \]
\[ \text{ends}(V, I). \]

\[ \text{ends}(A, I + 1) \leftarrow \text{occurs}(A, I). \]

together with some auxiliary axioms initiate a sequence \( S \) of actions and sustain it until completion.
Testing the Theory: Project Halo

Project Halo is a research effort by Vulcan Inc. towards the development of a Digital Aristotle: a reasoning system capable of answering novel questions and solving advanced problems in a broad range of scientific disciplines and related human affairs.

The project focuses on creating two primary functions: a tutor capable of instructing and assessing students in those subjects, and a research assistant with broad, interdisciplinary skills to help scientists and others in their work.
Example: Cell Cycle

We assume that

(a) Cell Cycle consists of three consecutive steps: interphase, mitosis and cytokinesis

(b) We are given a hierarchy of classes of parts, e.g. nucleus is a part of a cell, chromosome is a part of nucleus, etc.

We’ll be interested in the number of different parts present in the environment during different stages of the cell cycle.
Basic Cell Cycle in \textit{ALM}

\textbf{module} \textit{basic\_cell\_cycle}

\textbf{sort} declarations

\textit{classes\_of\_parts} \% \textit{c\_o\_p} : \textbf{sort}
\textit{numbers} : \textbf{sort}

\textbf{fluent} declarations

\textit{father}(\textit{c\_o\_p}, \textit{c\_o\_p}) : \textbf{static fluent}
\textit{root}(\textit{c\_o\_p}) : \textbf{static fluent}
\textit{num}(\textit{c\_o\_p}, \textit{c\_o\_p}, \textit{numbers}) : \textbf{inertial fluent}
\% \textit{num}(C_1, C_2, N) – \textit{N} is the number of parts of class \textit{C}_1 in one part of class \textit{C}_2.
Basic Cell Cycle in $\mathcal{ALM}$

axioms

\[-\text{num}(C_1, C_2, N_2) \text{ if } \text{num}(C_1, C_2, N_1), \]
\[N_1 \neq N_2.\]

\[\text{num}(C_3, C_1, N) \text{ if } \text{father}(C_1, C_2),\]
\[\text{num}(C_2, C_1, N_1),\]
\[\text{num}(C_3, C_2, N_2),\]
\[C_3 \neq C_2,\]
\[N = N_1 * N_2.\]

end of num

$\text{prevented}_\text{dupl}(c_\text{o}_p)$ : inertial fluent
Basic Cell Cycle in $ALM$

**action declarations**

`duplicate : action`

**attributes**

`class : classes_of_parts`

**axioms**

`duplicate causes num(C_1, C_2, N_2)`

if `class = C_1`,

`father(C_2, C_1),`

`num(C_1, C_2, N_1),`

`N_2 = 2 * N_1.`

**impossible duplicate if** `class = C`,

`prevented_dupl(C').`

**end of duplicate**
Basic Cell Cycle in ALM

\textit{split} : \textit{duplicate}

axioms

\textit{split} \textit{causes} \textit{num}(C_1, C_2, N_2)

\textbf{if} \hspace{1em} \textit{class} = C_2,

\textit{father}(C_2, C_1),

\textit{num}(C_1, C_2, N_1),

N_1 \neq 0,

N_2 = N_1/2.

\textbf{end of split}
prevent_duplication : action

attributes

class : classes_of_parts

axioms

prevent_duplication causes prevented_dupl(C)

if class = C.

end of prevent_duplication
Representing Sequences of Actions

module sequences

sort declarations

  elements : sort
  sequences : sort
  numbers : sort

fluent declarations

  component(elements, numbers, sequences) : static fluent
  length(numbers, sequences) : static fluent

with axioms specifying the functional character of these relations.
Reasoning About Cell Cycle

Various system descriptions of ALM specifying this process on different levels of granularity will contain the basic_cell_cycle and sequence modules and will differ from each other only by their structure.

First, we consider a model in which cell cycle is viewed as a sequence of three elementary actions: interphase, mitosis, and cytokinesis. We also limit our domain to cells contained in an experimental environment that we will call sample.
Cell Cycle 1

The first refinement of Cell Cycle will include modules *basic_cell_cycle* and *sequences*, and:

**structure of** *cell_cycle*(1)

**sorts**

*sample, cell, nucleus in classes_of_parts*

*cell_cycle in sequences*

*interphase, mitosis, cytokinesis in elements*

**actions**

**instance** *interphase : action*

**instance** *mitosis : duplicate*

\[\text{class := nucleus}\]

**instance** *cytokinesis : split*

\[\text{class := cell}\]
Cell Cycle(1)

statics

father(sample, cell).
father(cell, nucleus).

component(interphase, 1, cell_cycle).
component(mitosis, 2, cell_cycle).
component(cytokinesis, 3, cell_cycle).
length(3, cell_cycle).
Reasoning about Cell Cycle

Suppose now that our initial sample consists of one cell with one nucleus.

We would like to know the number of cells and nuclei in the sample after the end of cell cycle. The answer can be obtained from the answer set of a program consisting of the ASP translation of the cell_cycle(1) system description and the domain history.
Reasoning about Cell Cycle

The history, $\mathcal{H}_1$, is written as:

$\text{observed}(\text{num}(\text{cell, sample}, 1), \text{true}, 0)$.

$\text{observed}(\text{num}(\text{nucleus, cell}, 1), \text{true}, 0)$.

$\text{intend}(\text{cell_cycle}, 0)$.

The answer set will contain the last step, 3, and facts

$\text{holds}(\text{num}(\text{cell, sample}, 2), 3)$

$\text{holds}(\text{num}(\text{nucleus, cell}, 1), 3)$

$\text{holds}(\text{num}(\text{nucleus, sample}, 2), 3)$

At the end the sample contains two cells with one nucleus each.
Reasoning about Cell Cycle

Suppose now we learned that:

In some organisms mitosis occurs without cytokinesis occurring.

and want to know how many nuclei are contained in a cell from the sample at the end of the cell cycle.

To answer the question we simply expand the history by

\(-\text{happened}(\text{cytokinesis}, I)\)

for every step $I$. The corresponding answer set will now contain:

\(\text{holds}(\text{num}(\text{cell, sample}, 1), 2)\)

\(\text{holds}(\text{num}(\text{nucleus, cell}, 2), 2)\)

\(\text{holds}(\text{num}(\text{nucleus, sample}, 2), 2)\)
Second Refinement of Cell Cycle

Let us now consider the following question Q12.15:

A researcher treats cells with a chemical that prevents DNA synthesis. This treatment traps the cells in which part of the cell cycle?

To answer this question the system will need to know more about the structure of the cell and that of the interphase and mitosis.

The second refinement of Cell Cycle provides this additional knowledge.
Knowledge for Second Refinement

Additional cell components: the chromosomes inside the nucleus, the chromatids that are part of the chromosomes, and the DNA inside the chromatids.

The interphase is a sequence \([g_1, s, g_2]\) where \(g_1\) and \(g_2\) are elementary actions and \(s\) is a sequence of two elementary actions: DNA synthesis, and the creation of sister chromatids.

Mitosis is a sequence of five actions: prophase, prometaphase, metaphase, anaphase, and telophase.

The treatment of the cells with the chemical is represented by exogenous action that prevents the duplication of the DNA.
Cell Cycle(2)

structure of cell_cycle(2)

sorts

sample, cell, nucleus in classes_of_parts
chromosome, chromatid, dna in classes_of_parts
cell_cycle, interphase, s, mitosis in sequences
interphase, mitosis, cytokinesis in elements
g1, s, g2, dna_synthesis, prophase in elements
sister_chromatids, prometaphase in elements
metaphase, anaphase, telophase in elements
Cell Cycle(2)

actions

instance \( g_1 \) : action

instance \( dna\hspace{1pt}_\text{synthesis} \) : duplicate

class := dna

instance \( sister\hspace{1pt}_\text{chromatids} \) : split

class := chromatid

instance \( g_2 \) : action

instance \( prophase \) : action

instance \( prometaphase \) : action

instance \( metaphase \) : action

instance \( anaphase \) : split

class := chromosome

instance \( telophase \) : split

class := nucleus
Cell Cycle(2)

% actions continued

instance cytokinesis : split
    class := cell

instance treatment : prevent_duplication
    class := dna

statics

father(sample, cell).
father(cell, nucleus).
father(nucleus, chromosome).
father(chromosome, chromatid).
father(chromatid, dna).

component(interphase, 1, cell_cycle).
component(mitosis, 2, cell_cycle).
component(cytokinesis, 3, cell_cycle).
length(3, cell_cycle).
Cell Cycle(2)

component(g1, 1, interphase).
component(s, 2, interphase).
component(g2, 3, interphase).
length(3, interphase).

component(dna_synthesis, 1, s).
component(sister_chromatids, 2, s).
length(2, s).

component(prophase, 1, mitosis).
component(prometaphase, 2, mitosis).
component(metaphase, 3, mitosis).
component(anaphase, 4, mitosis).
component(telophase, 4, mitosis).
length(5, mitosis).
Reasoning about Cell Cycle(2)

We can now capture the scenario in question Q12.15 via the following history $\mathcal{H}_2$

\begin{align*}
\text{observed}(\text{num}(\text{cell}, \text{sample}, 1), \text{true}, 0). \\
\text{observed}(\text{num}(\text{nucleus}, \text{cell}, 1), \text{true}, 0). \\
\text{observed}(\text{num}(\text{chromosome}, \text{nucleus}, 46), \text{true}, 0). \\
\text{observed}(\text{num}(\text{chromatid}, \text{chromosome}, 1), \text{true}, 0). \\
\text{observed}(\text{num}(\text{dna}, \text{chromatid}, 1), \text{true}, 0). \\
\text{intend}(\text{cell\_cycle}, 0). \\
\text{happened}(\text{treatment}, 0).
\end{align*}
Reasoning about Cell Cycle(2)

To answer our question we define a relation \textit{trapped}:

\begin{verbatim}
ended(V) :- ends(V, I), step(I).
trapped(V1) :- component(V1, K, S),
                component(V2, K+1, S),
                ended(V1),
                not ended(V2).
\end{verbatim}

and add this definition to the ASP encoding. The answer set of the resulting program will contain \textit{trapped}(g1), where g1 is the answer to question Q12.15.
Conclusions

• The characteristic features of $\mathcal{ALM}$ are its closeness to logic programming and the functional character of its modules. This is achieved by the use of $\mathcal{AL}$ and the separation between general uninterpreted declarations and their domain dependent interpretations.

• System descriptions of $\mathcal{ALM}$ can be used together with fairly sophisticated histories of the domain to allow non-trivial reasoning about past and future.

This includes planning, diagnostics, and hypothetical reasoning.
Conclusions

• This reasoning is reduced to computing answer sets of logic programs. This allows the use of efficient answer set solvers and other inference engines sound w.r.t. answer set semantics.

• Reasoning can be proven correct w.r.t. an ALM based model of a dynamic system.

• The use of ALM facilitates the creation of library modules and supports the reuse of knowledge. In particular, ALM allows definitions of fluents and actions in terms of other fluents and actions.
Future Work

• $\mathcal{ALM}$ is work in progress. More experience of its use is needed to fix some details.

We also plan to

• automate the translation of $\mathcal{ALM}$ into logic programs.

• study the mathematical properties of $\mathcal{ALM}$.

• use an $\mathcal{ALM}$ based system to build KR libraries.