On the Relationship between Two Modular Action Languages:
A Translation from MAD into ALM

Daniela Inclezan
Miami University

Introduction

Action languages are high-level declarative languages dedicated to the concise and elegant representation of dynamic systems.

Dynamic system – system that can be represented by a transition diagram whose nodes correspond to possible states and arcs are labeled by actions.

Action languages like $\mathcal{B}$ ($\mathcal{AL}$), $\mathcal{C}$, $\mathcal{C}+$ etc. were introduced to address important problems in the field of RAC.

Introduction (cont.)

A next challenge: the creation of *libraries of commonsense knowledge* and *large knowledge bases* about dynamic domains.

- Traditional action languages lack the means for the **reuse** and **structuring** of knowledge needed to address this problem.
- Modular action languages
  - MAD (Lifschitz and Ren 2006; Erdogan and Lifschitz, 2006) and
  - ALM (Gelfond and Inclezan 2009; Inclezan and Gelfond 2015)

  were introduced to address this problem.

Introduction (cont.)

MAD and ALM have **common goals**

- How to represent actions, especially in terms of previously defined actions? (e.g., *carry* is defined in terms of *move* as *to move while holding*).

They use different **mechanisms for the reuse** of knowledge:

- MAD: *import statements* and *renaming clauses*
- ALM: objects, including actions, are organized in a sort hierarchy defined using the *specialization construct*.

Goal of This Work

Study the relationship between MAD and ALM, and especially that between their **mechanisms for the reuse of knowledge**.

In this paper:

1. Provide a translation from MAD into ALM.
2. Determine a class of MAD action descriptions for which the translation produces transition diagrams isomorphic to the original ones, modulo the common vocabulary.

Language ALM

Language ALM

- Dynamic system described by a **system description**:
  - **Theory** – one or more **modules** on a common theme organized in hierarchy
  - **Structure** – interpretation of symbols in the theory.
- **Module:**
  - a collection of declarations of sorts and functions, together with a set of axioms.
  - organizes knowledge into smaller reusable pieces of code.
- Actions are defined in terms of previously defined actions via the **specialization construct**.

ALM: Theory/ Module Example

sort declarations

$points, things$ :: **universe**

$agents, carriables$ :: $things$

$move$ :: **actions**

attributes

$actor : agents$

$origin, dest : points$

$carry$ :: $move$

attributes

$carried\_thing : carriables$

ALM: Module Example (cont.)

function declarations

fluents basic

total $loc\_in : things \rightarrow points$

axioms

$occurs(X)$ **causes** $loc\_in(A) = D$ **if** $instance(X, move),$

$actor(X) = A,$

$dest(X) = D.$

…

ALM: Structure Example

structure

  instances

  $john, bob$ **in** $agents$

  $london, paris$ **in** $points$

  $go(A, P)$ **in** $move$

    $actor = A$

    $dest = P$

ALM: Informal Semantics

- Translate system descriptions of ALM into programs of ASP{f} (Balduccini 2013).
- Answers sets of such programs define the states and transitions of transition diagrams.
- Translation example:

$$
\begin{aligned}
&occurs(X)\ \textbf{causes}\ loc\_in(A) = D\ \textbf{if}\ instance(X, move),\\
&\qquad actor(X) = A,\\
&\qquad dest(X) = D.
\end{aligned}
$$

becomes

$$
\begin{aligned}
loc\_in(A, I+1) = D \ \leftarrow \ & instance(X, move),\\
& occurs(X, I),\\
& actor(X) = A, dest(X) = D.
\end{aligned}
$$

Language MAD

Language MAD

- Dynamic system described by an **action description**:
  - Declarations of **sorts** and subsort relations
  - One or more **modules**
- A **module** consists of:
  - declarations of objects, actions, fluents, and variables;
  - import statements; and
  - axioms.
- Actions are represented using terms, not sorts.
- **Import** statements allow the renaming of sorts, fluents, and actions (and thus defining special case actions).

MAD: Action Description Example

Encoding extracted from (Erdogan 2008).

sorts

$Domain; Range; Thing; Place;$

module $ASSIGN;$

actions $Assign(Domain, Range);$

fluents $Value(Domain) : simple(Range);$

variables $x : Domain;\ y : Range;$

axioms

inertial $Value(x);$

exogenous $Assign(x, y);$

$Assign(x, y)$ **causes** $Value(x) = y;$

MAD: Action Description (cont.)

module $MOVE$;

actions $Move(Thing, Place)$;

fluents $Location(Thing) : simple(Place)$;

variables $x : Thing; y : Place$;

import $ASSIGN$;

$Domain$ **is** $Thing$;

$Range$ **is** $Place$;

$Value(x)$ **is** $Location(x)$;

$Assign(x, p)$ **is** $Move(x, p)$;

axioms

…

MAD: Action Description (cont.)

module MB;

objects $Monkey : Thing$;

$P_1, P_2 : Place$;

actions $Walk(Place)$;

variables $p : Place$;

import $MOVE$;

$Move(Monkey, p)$ **is** $Walk(p)$;

MAD: Informal Semantics

- Flatten action descriptions and translate them into $\mathcal{C}+$.
- Flattening process:
  - Replace sort names by new names.
  - Add a prefix of the type "In." to variables, renamed fluents, and renamed actions.
  - Add axioms to capture the renaming of fluents and actions.
- Example (axioms added when flattening):

$\text{I1.}\,Assign(\text{I2.I1.}x, \text{I2.I1.}y)$ **causes** $\text{I1.}\,Value(\text{I2.I1.}x) = \text{I2.I1.}y;$

$\text{I1.}\,Value(\text{I2.}x) \equiv Location(\text{I2.}x);$

$\text{I1.}\,Assign(\text{I2.}x, \text{I2.}p) \equiv \text{I2.}\,Move(\text{I2.}x, \text{I2.}p);$

$\text{I2.}\,Move(Monkey, p) \equiv Walk(p);$

Translation

Translation: Main Challenges

1. **Actions** are represented using terms of MAD. In ALM, there are action (sub)sorts and action instances.
2. **Sorts** can be renamed in MAD; renamed sorts are synonymous to the original ones. ALM can only describe special cases.
3. **Fluents** can be renamed in MAD. There is no equivalent concept in ALM.
4. **Objects** can be defined in MAD modules. In ALM, only very general objects may be included in modules.

Translation: Restrictions

I limited myself to action descriptions of MAD

- whose import statements and axioms satisfied certain syntactic constraints and
- only contained simple *inertial* fluents and *exogenous* actions.

Let us call such action descriptions *simple*.

Translation Key Points: Sorts

- Translate renaming clauses for sorts via the specialization construct.

$$\begin{array}{l} \textbf{import}\ ASSIGN; \\ \quad Domain\ \textbf{is}\ Thing; \\ \quad Range\ \textbf{is}\ Place; \end{array}$$

$$\begin{array}{lcl} thing & :: & domain \\ place & :: & range \end{array}$$

Translation Key Points: Actions (1)

- Translate a MAD action into an action subsort of ALM if
  - there are axioms about it or
  - it appears on the RHS of a renaming clause that does not contain objects.
- Add attributes to represent parameters.
- Modify axioms about actions to reference the added attributes.

Example: Actions (1)

$Assign(Domain, Range);$

$Assign(x, y)$ **causes** $Value(x) = y;$

$assign$ :: **actions**
attributes
$attr1\_assign$: $domain$
$attr2\_assign$: $range$

$occurs(A)$ **causes** $value(X) = Y$ **if** $instance(A, assign),$
$attr1\_assign(A) = X,$
$attr2\_assign(A) = Y.$

Example: Actions (1)

$Move(Thing, Place);$

import $ASSIGN;$

$\quad Assign(x, p)$ **is** $Move(x, p);$

$move :: assign$

$\quad$ **attributes**

$\quad\quad attr1\_move : thing$

$\quad\quad attr2\_move : place$

$$attr1\_assign(A) = X \textbf{ if } instance(A, move), attr1\_move(A) = X.$$

$$attr2\_assign(A) = X \textbf{ if } instance(A, move), attr2\_move(A) = X.$$

Translation Key Points: Actions (2)

- Translate all other MAD actions (given the mentioned restrictions) as instances of actions.

$$Walk(Place);$$

$$\textbf{import } MOVE;$$
$$\quad Move(Monkey, p) \textbf{ is } Walk(p);$$

$$walk(P) \textbf{ in } move$$
$$\quad attr1\_move = monkey$$
$$\quad attr2\_move = P$$

Translation Key Points: Fluents

- Translate renaming clauses for fluents via state constraints.

$Location(Thing) : simple(Place);$

import $ASSIGN$;
  $Value(x)$ **is** $Location(x)$;

fluents basic total $location : thing \rightarrow place$

$location(X_1) = X_2$ **if** $value(X_1) = X_2$.
$value(X_1) = X_2$ **if** $location(X_1) = X_2$.

Issue 1

module MB;
  objects $Box, Monkey : Thing;\ P_1, P_2 : Place$;
  actions $PushBox(Place)$;
  variables $p : Place$;

  import $MOVE$;
    $Move(Monkey, p)$ **is** $PushBox(p)$;
  import $MOVE$;
    $Move(Box, p)$ **is** $PushBox(p)$;

$pushbox(P)$ **in** $move$
  $attr1\_move = monkey$
  $attr1\_move = box$
  $attr2\_move = P$

Issue 1: Solution

- Expand attributes by adding a new parameter (their original range) and making them range over Booleans.

$$
\begin{aligned}
&pushbox(P)\ \mathbf{in}\ move\\
&\quad attr1\_move(monkey) = true\\
&\quad attr1\_move(box) = true\\
&\quad attr2\_move(P) = true
\end{aligned}
$$

Issue 2

Two fluents

$$Location(Thing) : simple(Place);$$
$$Support(Thing) : simple(Supporter);$$

import $ASSIGN$;
$Value(x)$ **is** $Location(x)$;
import $ASSIGN$;
$Value(t)$ **is** $Support(t)$;

$location(X_1) = X_2$ **if** $value(X_1) = X_2$.
$value(X_1) = X_2$ **if** $location(X_1) = X_2$.
$support(X_1) = X_2$ **if** $value(X_1) = X_2$.
$value(X_1) = X_2$ **if** $support(X_1) = X_2$.

value is no longer a function

Issue 2: Solution

- Expand *renamed* functions by adding a new parameter (the original range) and making them range over Booleans.

$$place, supporter :: range$$

$$value : domain \times range \rightarrow booleans$$
$$location : thing \rightarrow place$$
$$support : thing \rightarrow supporter$$

$$value(X,Y) \textbf{ if } location(X) = Y.$$
$$location(X) = Y \textbf{ if } value(X,Y).$$
$$\neg value(X,Y) \textbf{ if } location(X) \neq Y.$$
$$\neg value(X,Y) \textbf{ if } value(X,Z), Y \neq Z,$$
$$instance(X,place), instance(Y,place).$$

This works if sorts *place* and *supporter* have disjoint interpretations.

Properties of the Translation

Properties of the Translation

Goal:

- Find a class of MAD action descriptions whose ALM translations define transition diagrams isomorphic to the original ones, modulo the common vocabulary.

Preliminary Definition

- Let AD be a MAD action description and $\alpha(AD)$ its corresponding ALM translation.

A function of AD

$$f(s_1, \dots, s_n) : \langle type \rangle (s_{n+1})$$

is **well-defined** if for every interpretation I of $\alpha(AD)$ and every pair of functions

$$g(z_1, \dots, z_n) : \langle type \rangle (z_{n+1}) \text{ and}$$
$$h(c_1, \dots, c_n) : \langle type \rangle (c_{n+1})$$

such that both g and h are special cases of f,

$$\exists k, 1 \leq k \leq n, \text{ such that } I(z_k) \cap I(c_k) = \emptyset.$$

Proposition (Informal)

- If all functions of a *simple* action description are *well-defined* then its transition diagram will be isomorphic to the transition diagram defined by its ALM translation, modulo the following differences:

States:	MAD fluents contain additional prefixes ALM states contain some expanded fluents and predefined statics
Transitions:	MAD transition will be labeled by additional actions (the actions that were renamed)

Conclusions and Future Work

- Proposed a translation from MAD into ALM and a class of action descriptions for which the translation is adequate.
  - This allows for libraries of knowledge developed in MAD to be seamlessly combined with knowledge modules written in ALM.
- Result: a better understanding of the relationship between the constructs for the reuse of knowledge and the description of actions as special cases of other actions.
- Future work: study other constructs of MAD.