Theory of Activities

May 4, 2018

1 The Physical Domain

The domain has four rooms located side by side ($library$, $kitchen$, $office1$ and $office2$) and connected. The robot rob_1, can move from one room to the next. A *secure* room can be *locked* or *unlocked*. The robot cannot move to or from a locked room; it can *unlock* a locked room. The domain objects can be located in any of the rooms. The robot can *pickup* an object if is in the same location as the object, it can *put_down* an object that it is holding and it can only hold one object at a time. There are two exogenous actions, one that can change the location of any object, and one that can lock a secure room. The agent may not be aware of these exogenous action when they happen. We are also going to define three different defined fluents, two of which we will be using as possible goals.

1.1 AL

Sorts:

$$secure\_room = \{library\}.$$
$$room = secure\_room\ + \{kitchen, office_1, office_2\}.$$
$$robot = \{rob_1\}.$$
$$book = \{book_1, book_2\}.$$
$$object = book.$$
$$thing = object\ +\ robot.$$

Static relations:

$$next\_to(office_1, office_2).$$

$$next\_to(kitchen, office_1).$$
$$next\_to(library, kitchen).$$

Inertial fluents:

$$inertial\_fluent = loc(thing, room) + in\_hand(robot, object) + locked(secure\_room).$$

Possible goals:

$$possible\_goal = tidy\_all(room) + tidy\_book(book, room).$$

Defined fluents:

$$defined\_fluent = possible\_goal + missing\_book(room).$$

Robot actions:

$$move(robot, room).$$
$$pickup(robot, object).$$
$$put\_down(robot, object).$$
$$unlock(robot, secure\_room).$$

Exogenous actions:

$$exo\_move(object, room).$$
$$exo\_lock(secure\_room).$$

Causal Laws:

$$\begin{array}{rcl} move(R, L) & \textbf{causes} & loc(R, L) \\ pickup(R, O) & \textbf{causes} & in\_hand(R, O). \\ put\_down(R, O) & \textbf{causes} & \neg in\_hand(R, O). \\ unlock(R, L) & \textbf{causes} & \neg locked(L). \\ exo\_lock(L) & \textbf{causes} & locked(L). \\ exo\_move(O, L) & \textbf{causes} & loc(O, L). \end{array}$$

State Constraints:

$$next\_to(L1, L2) \quad \textbf{if} \quad next\_to(L2, L1).$$

$$\begin{aligned}
\neg loc(T, L2) \quad &\textbf{if} \quad loc(T, L1),\ L1 \neq L2. \\
loc(O, L) \quad &\textbf{if} \quad loc(R, L),\ in_hand(R, O). \\
\neg in_hand(R, O1) \quad &\textbf{if} \quad in_hand(R, O2),\ O1 \neq O2.
\end{aligned}$$

Executability Conditions:

$$\begin{aligned}
\textbf{impossible} \quad move(R, L) \quad &\textbf{if} \quad loc(R, L). \\
\textbf{impossible} \quad move(R, L2) \quad &\textbf{if} \quad loc(R, L1),\ \neg next_to(L1, L2). \\
\textbf{impossible} \quad move(R, L2) \quad &\textbf{if} \quad loc(R, L1),\ locked(L1). \\
\textbf{impossible} \quad move(R, L) \quad &\textbf{if} \quad locked(L). \\
\textbf{impossible} \quad unlock(R, L) \quad &\textbf{if} \quad \neg locked(L). \\
\textbf{impossible} \quad unlock(R, L1) \quad &\textbf{if} \quad loc(R, L2),\ \neg next_to(L2, L1),\ L2 \neq L1. \\
\textbf{impossible} \quad put_down(R, O) \quad &\textbf{if} \quad \neg in_hand(R, O). \\
\textbf{impossible} \quad pickup(R, O1) \quad &\textbf{if} \quad in_hand(R, O2). \\
\textbf{impossible} \quad pickup(R, O) \quad &\textbf{if} \quad loc(R, L1),\ loc(O, L2),\ L1 \neq L2. \\
\textbf{impossible} \quad exo_move(O, L) \quad &\textbf{if} \quad loc(O, L) \\
\textbf{impossible} \quad exo_move(O, L) \quad &\textbf{if} \quad locked(L). \\
\textbf{impossible} \quad exo_move(O, L2) \quad &\textbf{if} \quad loc(O, L1), locked(L1). \\
\textbf{impossible} \quad exo_move(O, L) \quad &\textbf{if} \quad in_hand(R, O). \\
\textbf{impossible} \quad exo_lock(L) \quad &\textbf{if} \quad locked(L).
\end{aligned}$$

1.2 The Theory of Activities

In our ToA domain of our agent will also have a list of *possible goals* and one of them will be selected. The agent will need to specify one or more *activities* that would achieve the selected goal. He may have a list or different existing activites. If there are existing activities that achieve the goal, the agent will choose and return one of those activites. If the agent cannot find a successfull existing *activity*, he will create and return one. If he cannot use or create a successfull *activity*, the goal will be considered futile.

An *activity* will be represented by a triple consisting on *name*, *plan* and *goal*. A *name* is a unique identifier used to refer to the *activity*, and a plan is a sequence of agent actions, which will lead to the realisation of the *goal*.

We limit the names of activities to a collection of integers ($1 \ldots max\_name$), the length of plans to a maximum length ($1 \ldots max\_len$). The fluents of the physical environment that may serve as a *goal* are those of the sort *possible_goal*.

In order to create the Action Language for the new domain, we will 1-adapt and 2-extend the original Action Language for the physical domain. We will adapt it by re-defining the sort $inertial\_fulent$ as $physical\_inertial\_fluens$, $defined\_fluents$ as $physical\_defined\_fluents$, and the sort $agent\_action$ as $physical\_agent\_action$. We will define the following new sorts:

- A sort $activity\_name = 1, \ldots, max\_name$ to represent the name of an *activity*.
- A sort $index = \{-1, 0, max\_len\}$ to represent the index of an action as part of an *activity*.
- A sort $mental\_agent\_action = \{select\_activity(activity\_name)\}$ to represent the action of choosing and activity.
- A sort $mental\_inertial\_fluent = \{current\_aciton\_index(activity\_name, index)\}$ to represent the current (mental) state of an *activity*.
- A sort $mental\_defined\_fluent = \{next\_action(activity\_name, action)\}$

We also the following relations that give shape to the concept of *activity*.

$$activity\_component(activity\_name, index, physical\_agent\_action).$$
$$activity\_length(activity\_name, index).$$
$$activity\_goal(activity\_name, possible\_goal).$$

We create a hierarchical structure of $fluents$ and *actions* as follows:

$$\begin{aligned}
&inertial_fluent = physical_inertial_fluent + mental_inertial_fluent,\\
&fluent = defined_fluent + inertial_fluent,\\
&agent_action = mental_agent_action + physical_agent_action,\\
&action = agent_action + exo_action.
\end{aligned}$$

As well as the previous statements included in the physical domain, we will include:

Causal Laws:

$$select\_activity(AN) \quad \textbf{causes} \quad current\_action\_index(AN, 0). \tag{1}$$

$$\begin{aligned}
PAA \quad \textbf{causes} \quad current_action_index(AN, K+1) \quad \textbf{if} \quad & next_action(AN, PAA),\\
& current_action_index(AN, K),\\
& activity_component(AN, K+1, PAA).
\end{aligned} \tag{2}$$

State Constraints:

$$\begin{aligned}
\neg current_action_index(AN, K1) \quad \textbf{if} \quad & current_action_index(AN, K2),\\
& K1 \neq K2.
\end{aligned} \tag{3}$$

$$\begin{aligned} next\_action(AN, PAA) \quad \textbf{if} \quad & current\_action\_index(AN, K), \\ & activity\_component(AN, K+1, PAA). \end{aligned} \tag{4}$$

$$\begin{aligned} tidy\_book(B, R) \quad \textbf{if} \quad & loc(B, R), \neg in\_hand(B). \\ missing\_book(R) \quad \textbf{if} \quad & \neg tidy\_book(B, R). \\ tidy\_all(R) \quad \textbf{if} \quad & not\ missing\_book(R). \end{aligned} \tag{5}$$

$$\begin{aligned} \neg selected\_goal(G1) \quad \textbf{if} \quad & selected\_goal(G2), \\ & G1 \neq G2. \end{aligned} \tag{6}$$

Initial State Constraint:

$$current\_action\_index(AN, -1) \tag{7}$$

2 The Architecture: Reasoning Tasks and Behaviour.

2.1 Introducing new relations

The axioms that need to be added to the ASP program also involve the following relations:

- $impossible(A, I)$ means that action A is impossible at step I.
- $candidate(AN)$ means that AN is a candidate for the next activity to be started to achieve the selected goal.
- $existing\_candidate(AN)$ means that AN is an existing candidate for the activity to achieve the selected goal.
- $new\_candidate(AN)$ means that AN is newly created candidate for the activity to achieve the selected goal.
- $has\_component(AN, K)$ means that activity AN has a component at index K.
- $needs\_new\_activity$ flag that means that no successful existing activity has been found in the previous run.
- $next\_available\_name$ will hold the next number that can be used as a name for a newly created activity.

2.2 Translating AL to ASP

The following steps should be followed in order to translate the AL description into an ASP program.

For every causal law: a **causes** l **if** $p_0, \ldots, p_m$
Tthe ASP contains: $holds(l, I+1)$:- $holds(p_0, I), \ldots, holds(p_m, I), occurs(a, I)$.

For every state constraint: l **if** $p_0, \ldots, p_m$
the ASP contains $holds(l, I)$:- $holds(p_0, I), \ldots, holds(p_m, I)$.

The ASP contains the CWA for defined fluents:
-$holds(F, I)$:- $\#defined\_fluent(F)$, *not* $holds(F, I)$.

For every executability condition: **impossible** a **if** $p_0, \ldots, p_m$
the ASP contains: $impossible(a, I)$:- $holds(p_0, I), \ldots, holds(p_m, I)$.
It also contains: -$occurs(A, I)$:- $impossible(A, I)$.

The ASP contains the inertia axioms:
$holds(F, I+1)$:- $holds(F, I)$, *not* -$holds(F, I+1)$.

-$holds(f, I+1)$:- -$holds(F, I)$, *not* $holds(F, I+1)$.

The ASP contains the CWA for actions:
-$occurs(A, I)$:- *not* $occurs(A, I)$.

Once translation using the above steps has been completed, the axioms in the following section will also need to be added to the ASP program.

2.3 Reasoning Axioms

Defining candidates:

The flag *need_new_candidate* will trigger the creating of a new candidate:

$$new\_candidate(AN) \leftarrow needs\_new\_activity. \tag{8}$$

$$\begin{aligned} existing\_candidate(AN) \leftarrow\ & activity\_goal(AN, G), \\ & selected\_goal(G), \\ & not\ new\_candidate(AN). \end{aligned} \tag{9}$$

$$\begin{aligned} candidate(AN) &\leftarrow new\_candidate(AN). \\ candidate(AN) &\leftarrow existing\_candidate(AN). \end{aligned} \tag{10}$$

Creating new activities:

$$\begin{aligned} activity\_goal(AN, G) \leftarrow\ & new\_candidate(AN), \\ & selected\_goal(G). \end{aligned} \tag{11}$$

$$\begin{aligned} activity\_component(AN, I, PAA) \leftarrow\ & new\_candidate(AN), \\ & occurs(select\_activity(AN), 0), \\ & occurs(PAA, I), \\ & 0 < I. \end{aligned} \tag{12}$$

$$\begin{aligned} \leftarrow\ & new\_candidate(AN), \\ & activity\_component(AN, K, PAA1), \\ & activity\_component(AN, K, PAA2), \\ & PAA1 \neq PAA2. \end{aligned} \tag{13}$$

$$\begin{aligned} has\_component(AN, K) \leftarrow\ & new\_candidate(AN), \\ & occurs(select\_activity(AN), 0), \\ & activity\_component(AN, K, C). \end{aligned} \tag{14}$$

$$\begin{aligned} activity\_length(AN, K) \leftarrow\ & new\_candidate(AN), \\ & occurs(select\_activity(AN), 0), \\ & has\_component(AN, K), \\ & not\ has\_component(AN, K + 1). \end{aligned} \tag{15}$$

Selecting candidates: If it is not impossible to select a candidate, it will be selected.

$$\begin{aligned} occurs(select\_activity(AN), 0) \leftarrow\ & candidate(AN), \\ & not\ impossible(select\_activity(AN), 0). \end{aligned} \tag{16}$$

It is impossible to select an activity at any step other than 0, it is impossible to select an activity if it is not a candidate, and it is impossible to select two different activities .

$$
\begin{array}{rcl}
impossible(select\_activity(AN1), 0) & \leftarrow & occurs(select\_activity(AN2), 0), \\
& & AN1 \neq AN2. \\
impossible(select\_activity(AN), 0) & \leftarrow & not\ candidate(AN). \\
impossible(select\_activity(AN), I) & \leftarrow & 0 < I. \\
impossible(select\_activity(AN), I) & \leftarrow & activity\_goal(AN, G), \\
& & holds(G, 0).
\end{array}
\tag{17}
$$

This rule ensures that the selected exisiting activity is the minimal activity that reaches the goal:

$$
\begin{array}{rcl}
occurs(PAA, I) & \stackrel{+}{\leftarrow} & existing\_candidate(AN), \\
& & occurs(select\_activity(AN), 0), \\
& & holds(next\_action(AN, PAA), I), \\
& & occurs(A, I-1), \\
& & 0 < I.
\end{array}
\tag{18}
$$

This rule ensures that the newly created activity is the minimal activity that reaches the goal:

$$
\begin{array}{rcl}
occurs(PAA, I) & \stackrel{+}{\leftarrow} & new\_candidate(AN), \\
& & occurs(select\_activity(AN), 0), \\
& & occurs(A, I-1), \\
& & 0 < I.
\end{array}
\tag{19}
$$

Planning module:

$$
\begin{array}{rcl}
success & \leftarrow & holds(G, I),\ selected\_goal(G). \\
& \leftarrow & not\ success.
\end{array}
\tag{20}
$$