A Simple Concurrent Action Formalism

Marc Toussaint

May 14, 2015

1 Introduction

There are many existing formalisms and approaches to 1) describe concurrent (multi-agent) action domains and problems (including PDDL level 4); 2) to reduce such domains to normal Markovian (or semi-Markov) domains; and 3) solve the resulting temporal *planning* problems.

I see the situation as follows: Basic research was already done more than a decade ago. Now, the issues are rather:

- What is an efficient formalism that allows us to integrate it with our existing relational RL and robot control frameworks?
- What is an approach to also compute *posteriors* (e.g., for anticipation) rather than "only" solve (usually deterministic) planning problems
- What is a formulation that allows us to leverage Monte-Carlo Tree Search methods (UCT)?

In the following I propose a simple approach. Questions w.r.t. this approach are:

- Is it equally general (can cope with the same range of domains) as other formalisms?
- What notions of optimality are there, also in the stochastic duration case?
- More thoroughly formalism also the stochastic case.
- Optimality and theory on MCTS within this framework.

2 From Actions to Initiation & Termination Decisions

The classical formulations still think of the "actions" as the decision variables. In the concurrent case, they lift this to joint actions, i.e., making decisions about action tuples. But this raises ugly isues when actions have variable or even stochastic durations. All that led to the ideas of temporal planning, etc.

Much easier is the following process definition. We first describe *deterministic domains*:

2.1 Initiation & termination operators

Assume we have n action symbols (activity would be a better word, but too long). For each action symbol a we have *two* operators (instead of one, as with standard action operators):

- The *initiation operator*

$$a_I(X): \text{ precond}(X) \rightarrow \texttt{go}(a, X) = d_a, \text{other\_effects}$$

- The *termination operator*

$$a_T(X): \quad \rightarrow \neg\texttt{go}(a, X), \text{other\_effects}$$

Both of these operators change the current relational state *instantly*.

The go-predicate is the core: A $\texttt{go}(a, X)$-predicate in the current state indicates that the action a with arguments X is currently active. Further, the go-predicate is *real-valued*! Its current value is the **time-to-go** until this activity will terminate and change the relational state. The initiation operator initialized the value of the go-predicate to the (here deterministic) duration d_a of that action.

If multiple initiation operators initiate multiple actions instantly at the same point in time, they're all concurrently active, perhaps with different times-to-go.

2.2 Initiation & wait decisions

The resulting *decision* process is Markovian: The process is a series of two types of decisions: *initiation decisions* and *wait decisions*.

Each initiation decision might refer to a different agent (indicated by the argment X to an action a). Initiation decisions are instantaneous, have no real-time duration. Therefore, although they "increase the step-counter" of the Markov Decision Process—or the depth of the search tree—real-time does not progress.

The wait decision [[refer to previous literature on these within sMDP]] is a unique thing, has no arguments X or a, and does not really refer to a particular agent—it rather expresses that *all agents decide not to initiate anything further in the current relational state*. Therefore, the wait decision progresses real-time *until the relational state changes*. In real-world, there may be different "reasons" the relational state changes:

- A currently active activity may return feedback, e.g. on its convergence, failure, termination, change-in-success-probability, anything. In all of these cases the activity just adds a predicate to the relational state encoding the novel information.
- A sensor signal might add and observation predicate to the relational state. (Actually, if one thinks of the activity of sensors also as an action, then this case is no different to the first.)

However, in model-based planning we assume to have a model of the world. Therefore we assume to know (at least in the deterministic case) at what time an activity will change the relational state. The wait decision is therefore realized by a very particular operator on the relational state, which is not expressed by a rule with preconditions and effects, but by the following little "procedure":

1. Find the go-predicate with the minimal time-to-go value d_{min}.
2. Decrement all go-predicate-values by d_{min}.
3. For all $\texttt{go}(a, X)$ predicates with $\texttt{go}(a, X) = 0$ call the *termination operator* $a_T(X)$

2.3 Redundancies in the decision tree

Consider the decision tree for this process. There is undesirable redundancy in this tree: When multiple initiation decisions are made, their order should not patter. (At least in the

classical formalisms they don't—in my formalisms there could be interesting preconditions to allow/disallow certain orders.) If the order does not matter, we can identify the resulting nodes as equal.

3 Example

Here is an example domain: the box assembly. The initial part simply declares a number of symbols which can later be used to formulate literals (nothing but tuples of symbols or variables, in this representation).

The initial state is a set (conjunction) of literals.

The terminal condition is a set (conjunction) of literals.

For every action symbol (activity) an initiation and termination operator is defined. Each operator first declares variables (arguments of the operator), the precondition as a set of literals, and the effect as a set of literals.

Note that for this example we had to introduce symbols to that allow us to indicate mutexes between activities, like busy (that hand is busy and can't do something else), etc. We played with this a lot and think that also the classical formulations are by no means better or easier: They have to introduce a lot of extra formalism to indicate various types of mutexes (eternal, etc). Here, all mutexing is represented in the preconditions of initiation operators.

[The file syntax is actually a hierarchical hypergraph syntax, with tuples being hypernodes/edges. We interally implement all first order logic operations (esp. computing feasible variable substitutions) on such a generalized graph representation.]

```
## Syntactic keywords
Terminate
#Rule

## activities
pickingup
positioning
screwing
releasing
busy
inhandNil

## basic predicates
table
on
wall
screw
ground
object
material
humR
humL
rob
hand
used
inPosition
inhand
fixed

## constants
Constant 67
Constant 71
Constant 72
Constant 76
Constant 77
Constant 78

## initial state
(screw 67)
(object 67)
(ground 71)
(material 71)
(object 71)
(wall 72)
(material 72)
(object 72)
(humR 76)
```

```
(hand 76)
(humL 77)
(hand 77)
(rob 78)
(hand 78)
(inhandNil 76)
(inhandNil 77)
(inhandNil 78)

### terminal state

terminal { (used 67) (inPosition 71) (inPosition 72) }

### RULES

Rule activate_pickingup {
     X, Y
     (pickingup X Y)! (hand X) (object Y) (inhandNil X) (busy X)! (busy Y)!
     effect { (go pickingup X Y)=2.1 (busy X) (busy Y) }
}

Term (Terminate pickingup) {
     X, Y
     effect { (go pickingup X Y)! (inhand X Y) (inhandNil X)! (busy X)! }
}

Rule activate_positioning {
     X, Y
     (positioning X Y)! (hand X) (object Y) (inhand X Y) (material Y) (inPosition Y)! (busy X)!
     effect { (go positioning X Y)=3.5 (busy X) }
}

Term (Terminate positioning) {
     X, Y
     effect { (go positioning X Y)! (inPosition Y)  (busy X)! }
}

Rule activate_releasing {
     X, Y
     (releasing X Y)! (hand X) (object Y) (inhand X Y) (material Y) (busy X)!
     effect { (go releasing X Y)=1.0 (busy X) }
}

Term (Terminate releasing){
     X, Y
     effect { (go releasing X Y)! (inhand X Y)! (inhandNil X) (busy X)! (busy Y)! }
}

Rule activate_screwing {
     X, Y, Z, U, V, W
     (screwing X Y Z U V W)! (hand X) (screw Y) (inhand X Y) (wall Z) (ground W) (inPosition Z) (inPosition W) (hand U) (inhand U Z) (hand V) (inhand V W) (fixed Z W)
     effect { (go screwing X Y Z U V W)=8.0 (busy X) (busy U) (busy V) (inPosition Z) (inPosition W) }
}

Term (Terminate screwing) {
     X, Y, Z, U, V, W
     effect { (go screwing X Y Z U V W)! (fixed Z W) (used Y) (inPosition Z) (inPosition W) (busy X)! (busy U)! (busy V)! (inhand X Y)! (inhandNil X) }
}
```

References

D. Aberdeen and O. Buffet. Concurrent probabilistic temporal planning with policy-gradients. In *ICAPS*, pages 10–17, 2007. URL http://www.aaai.org/Papers/ICAPS/2007/ICAPS07-002.pdf.

E. Beaudry, F. Kabanza, and F. Michaud. Planning with concurrency under resources and time uncertainty. In *ECAI*, pages 217–222, 2010. URL http://books.google.com/books?hl=en&lr=&id=cLqiEJVT3u8C&oi=fnd&pg=PA217&dq=%22planning+algorithm+queries+the+Bayesian+network+to+estimate%22+%22with+actions+concurrency+under+resources+and+time%22+%22to+this+date,+many+different+approaches+have+been%22+%22%5B11,+6%5D.+Others+include+the+Factory+Policy+Gradient%22+&ots=Go_vAIznes&sig=0pLTt3Es8OlENVRG5vKQRi64u1Q.

O. Buffet and D. Aberdeen. The factored policy-gradient planner. *Artificial Intelligence*, 173

(5-6):722–747, Apr. 2009. ISSN 00043702. doi: 10.1016/j.artint.2008.11.008. URL http://linkinghub.elsevier.com/retrieve/pii/S0004370208001859.

G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 4885–4890. IEEE, 2005. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1582935.

S. Hart and R. Grupen. Learning generalizable control programs. *Autonomous Mental Development, IEEE Transactions on*, 3(3):216–231, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5680948.

T. Luksch, M. Gienger, M. Muhlig, and T. Yoshiike. Adaptive movement sequences and predictive decisions based on hierarchical dynamical systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2082–2088, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6385651.

Mausam and D. S. Weld. Planning with durative actions in stochastic domains. *J. Artif. Intell. Res.(JAIR)*, 31:33–82, 2008. URL http://www.aaai.org/Papers/JAIR/Vol31/JAIR-3102.pdf.

K. Rohanimanesh and S. Mahadevan. Coarticulation: An approach for generating concurrent plans in markov decision processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 720–727. ACM, 2005. URL http://dl.acm.org/citation.cfm?id=1102442.

D. E. Smith and D. S. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI*, volume 99, pages 326–337, 1999. URL http://cs.tju.edu.cn/faculty/zyfeng/Course/AI/ISI/ijcai99-tgp.pdf.

H. a. L. Younes and R. G. Simmons. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS*, volume 4, page 325, 2004. URL http://www.aaai.org/Papers/ICAPS/2004/ICAPS04-039.pdf.