Relational Machine
Interfacing between robot activity control and relational learning and planning methods

M Toussaint

May 19, 2015

1 Purpose

What is described here plays a similar role to finite-state machines for action sequencing in robotics (like SMASH, RBO's hierarchical automaton), or also hierarchical RL frameworks (options, etc). The motivation for the specific *Relational Machine* approach is to combine two aims:

- Represent sequencing and cuncurrency of activities (i.e., options, durative actions) in a way that matches directly with formalisms for relational reinforcement learning, MCTS planning and other relational AI algorithms.
- At the same time provide a simple framework that flexibly allows us to design (also by hand and direct coding & scripting) general concurrent activity policies, including the specification of the activities (control modes) themselves.

2 The Relational Machine itself

Centralism: We assume a centralized representation of the relational state (or belief state). That is, even when describing multi-agent systems, we do not consider a decentralized state as in DEC-POMDPs, but assume that all sufficient knowledge for future decisions (on the AI level) are accumulated in a central state representation.

More specifically, we represent three things in a central "knowledge base" (see notes on FOL):

- The fluent[1] **relational state** is a set of grounded literals. These literals represent ongoing activities, current termination criteria, sensor information, belief predicates, etc in a relational manner. These literals are mostly symbolic, but can be annotated with arbitrary dictionaries.
- The **set of symbols** describes which objects are currently in the domain. Typically this is static, but might be also dynamic or subject to uncertainty (see our uncertainty-over-existance work). Also this set of symbols describes which predicates are available, which relate to activities (like control modes, sensor activities/services, etc) as well as descriptive state symbols, termination condition symbols, uncertainty symbols etc.

[1] changable, time-varying

- The **set of rules** may comprise general first-order clauses or (probabilistic) relational rules. These rules generally describe state transitions—however, as 'activity indicators' are part of the state they can play also play the role of a policy: For instance, to represent "$P(a|s)$" a rule may, conditional to the state, add an activity predicate a to the state. But rules can also model $P(s'|s, a)$ (as action operators) and thereby define the set of feasible decisions in s; or model $P(s'|s)$, i.e., state transitions that are usually triggered by the environment instead of the agent. Rules may also model the (expected) real-time duration of such transitions (see concurrent actions notes).

The Relational Machine needs to play two roles

- during real-time operation in interaction with real perceptional and control activities, where it controls how the activity state progresses, i.e., where it implements the logic of what activities are triggered/terminated when, etc.

- as a forward simulation model on the relational level, to enable planning (esp. MCTS planning). In this case there is no interaction with real sensors and motors. Instead the knowledge base needs to include rules that simulate how these activities would interact with the relational state.

2.1 Real-time operation

In real-time, the Relational Machine interacts only with **activities**. All activities always have access to the full relational state (esp. they receive changes) and may trigger changes to the relational state. Planning (or decision making in general) is just special activity detailed in the next section.

A **change of the relational state** can be caused by any of the following events:

- An activity explicitly modifies the relational state. For instance, an sensor activity indicates a new termination condition. Or a motor activity indicates convergence or time-out.

- The set of rules auto-progresses the state. For instance, the rules represent a policy of what to do given a novel termination condition: a respective activity is cancelled (deleted from the relational state) and a new activity added

That's basically it.

The relational state determines the "currently active activities" as follows: Every "active activity" must be identified with exactly one fact (=grounded literal in the relational state). For instance `(positionControl hand obj1){ rel=[0 0 0.2] PD=[.5, .9, .1, 10.] }` will acivate a position controller that aims to align the position of the `hand` frame with the relative position $(0, 0, 0.2)$ in the `obj1` frame following a desired PD approach behavior with time scale 0.5, daming ratio 0.9 and max vel/acc $(0.1, 10)$.

Whenever such a fact is part of the state, such a position controller is active. Whenever this fact is deleted from the state, the position controller becomes inactive (or is removed from the task list of the underlying operational space controller).

Note that the relational/symbolic aspect of such activities is captured by the literal itself (the FOL tuple), whereas additional parameters of the activity are **annotated with a dictionary** (technically, a subgraph).

2.2 Example activities

One might distinguish three types of activities:

- **Control activities** impose a control task, typically between two objects (hand-object). "Adding a control task" typically means to add a cost term in an operational space controller.

  Control activities may report back on their progress: e.g. convergence in the case of a PD-type feedback controller, time-out in the case of a trajectory following task, failure/stalling.

- **Sensor & perceptual activities** add/delete facts, e.g., contact with an object, perception/recognition of an object, etc.

- **Planning, reasoning, sensor processing activities** which are computational processes that, as a result, add information or a decision to the relation state.

However, it may be natural to define activities that are jointly control & sensor activities (that heavily interweave control and force feedback, for instance).

2.3 Off-line operation

For stochastic forward simulation of the state, as needed, e.g., in MCTS planning, the Relational Machine needs to simulate 'itself' as well as the state changes that would otherwise be triggered by real motor & sensor activities. Here we would not aim at realistic (physical) siulations of the motor & sensor activities[2], but rather leverage approximate, symbolic models of the state progression. Such models can be learned from previous real-world executions (as in model-based Relational RL), or hand-coded as a domain abstraction.

For offline operation we assume that the knowledge base includes rules that forward predict the state changes (including their real-time durations) that would otherwise be triggered by real activities.

[2] this is also an interesting option