Execution Monitoring of High-Level Programs.

Mikhail Soutchanski
Department of Computer Science
University of Toronto
http://www.cs.toronto.edu/~mes/index.html

1 A statement of interest.

My interest to interconnections between the hybrid systems and AI stems from 1) the well known perspective [3] that both planning and control address a similar problem: choosing actions to influence a process, based on some model of that process, and (2) from my research project on execution monitoring of high-level programs (programs specify plans to achieve certain goals as explained below). In my research, I look to an expressive and well studied knowledge representation framework for modeling processes called the situation calculus. The situation calculus is a predicate logic language for representing and reasoning about actions and properties of dynamically changing worlds. It is employed to model continuous processes and physical systems [8; 9; 4], to diagnose what happened to the system [7] and to provide the foundations for a logic programming language Golog [9; 5]. The version of the situation calculus that I use has been described in [9; 5], and elsewhere. I will not go over the language here except to note the basic components. All changes to the world are the result of named *actions*. According to the important idea of [10; 8], continuous processes influencing one or more parameters can be considered to have a regular behavior that does not change until an action is executed. A sequence of actions (history) is represented by a term called a *situation*. The constant S_0 is used to denote the *initial situation*, namely the empty history. Non-empty histories are constructed using a distinguished binary function symbol *do*; $do(\alpha(x), s)$ denotes the successor situation to s resulting from performing the action denoted by term $\alpha(x)$. There is a special predicate $Poss(\alpha(x), s)$ used to state that action $\alpha(x)$ is executable in situation s. Relations whose truth values vary from situation to situation are called *relational fluents*. Similarly, functions whose values vary from situation to situation are called *functional fluents*. In terms of the control theory (decision theory), fluents are state variables influenced by a sequence of control actions (sequence of decisions, respectively) performed so far.

To axiomatize the primitive actions and fluents of a domain of application, one must provide the following axioms:

1. Axioms describing the initial situation – what is true initially, before any actions have occurred. This is any finite set of sentences that mention only the situation term S_0, or that are situation independent.
2. Action precondition axioms, one for each primitive action. They characterize $Poss(A(\vec{x}), s)$: the conditions under which it is possible to execute action $A(\vec{x})$ in situation s. In addition to these, one must provide suitable unique names axioms for actions.
3. Successor state axioms, one for each fluent F, stating under what conditions the value of $F(\vec{x}, do(a, s))$ is determined as function of what holds in a previous situation s. These characterize effects of actions and also provide a solution to the frame problem [9; 5].

The set $\mathcal{D}$ of domain specific situation calculus axioms is employed to derive plans as explained below. Planning is known to be computationally intractable in general and is impractical for deriving complex behaviors involving hundreds, and possibly thousands of actions in applications characterized by hundreds of different fluents. For this reason, the University of Toronto Cognitive Robotics Group is pursuing a computer science perspective: reduce the reliance on *planning* for eliciting interesting behaviors, and instead provide the control system with *programs* written in a suitable high-level language, in our case, Golog or ConGolog. As presented in [6; 5] and extended in [1], Golog is a logic-programming language whose primitive actions are those of a background domain theory $\mathcal{D}$. Golog includes the following constructs: $\phi?$ – test the truth value of a situation calculus formula ϕ, $(\delta_1; \delta_2)$ – sequence of two programs, $(\delta_1 \mid \delta_2)$ – nondeterministic choice between programs, $\pi v.\delta$ – nondeterministic choice of argument to a program δ, as well as other constructs such as loops, conditionals and recursive procedures (see [1; 5] for details). A programmer may use any of these constructs to write a Golog program that constrains the search for a desirable plan (of course, if a Golog program is deterministic, then no search is required). It is the task of a Golog interpreter to figure out how to obtain a desirable plan (sequence of actions) as a side effect of executing a Golog program. In my research I rely on the single-step interpreter [1] that selects on each step of interpretation either a next primitive action for execution or a next test for evaluation. This interpreter is defined by means of two relations: *Trans* and *Final*; see [1] for details. Given a program δ and a situation s, *Trans*(δ, s, δ', s') tells us which is a possible next step in the computation, returning the resulting situation s' and the program δ' that remains to be executed. *Final*(δ, s) tells us whether δ can be considered final, that is whether the computation is completed (no program remains to be executed). I will not give here any axioms that provide operational transitional semantics for Golog constructs in terms of *Trans* and *Final*, because they are available in [1], but I note that these axioms characterize what one would normally expect. In the case of a primitive action a, $Trans(a, s, nil, do(a, s))$ is true iff $Poss(a, s)$ is true, where nil is the empty program (for any situation s $Final(nil, s)$); in the case of a test, $Trans(\phi?, s, nil, s)$ is true iff the situation calculus formula ϕ is true in s. The relation $Do(\delta, s, s')$ defined by means of *Trans** (the reflexive transitive closure of *Trans*) and *Final* specifies an interpreter of Golog programs as: $Do(\delta, s, s') \equiv \exists \delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$. In other words, $Do(\delta, s, s')$ holds iff it is possible to repeatedly single-step the program δ, obtaining a program δ' and a situation s' such that δ' can legally terminate in s'. A sequence of actions that leads from s to s' is a desired plan.

Thus, the interpreter derives a plan as a side-effect of executing a Golog program and whenever it selects a next primitive action A for execution it determines from a corresponding precondition axiom whether $Poss(A, S)$ is true or not in the current situation S, and whenever it selects a test ϕ, the evaluation of this test is also determined by axioms of the background domain theory $\mathcal{D}$. It is important to note that the logical (mental) representation of the world $\mathcal{D}$ does not contain information whether unpredictable changes actually occurred in the real world. The real world evolves according to its own incompletely known continuous (or discrete) laws and usually actions constituting a plan are not solely responsible for changes in the real world. Hence, the interpreter may not guarantee that truth values of relational fluents or values of functional fluents computed from $\mathcal{D}$ correspond to their values in the real world. For this reason, after selecting a primitive action for execution or evaluating a test condition, a high-level control module has to compare a mental world model with reality. If it does not notice any *relevant* discrepancies, then it executes the action in reality (or evaluates the test, respectively). Otherwise, the high-level control module (called the *monitor*) attempts to *recover* from unexpected discrepancies and then proceeds with the remaining part of the program or fails. The processes of interpreting and execution monitoring continues until the program reaches the final configuration or fails. It is convenient to consider all discrepancies as the result of sensory actions, exogenous with respect to our control system, and assume that the monitor observes all such actions. For example, to make sure that a next primitive action A (selected tentatively for execution) is possible in a current situation, the monitor has to sense values of fluents mentioned in the precondition axiom for A. In [2], the processes of interpreting and execution monitoring are characterized formally by a new predicate symbol $TransEM(\delta_1, s_1, \delta_2, s_2)$, describing a one-step transition consisting of a single $Trans$ step of program interpretation, followed by a process, called $Monitor$, of execution monitoring. The formal definition of $Monitor$ is parametric with respect to the two additional predicates $Relevant$ and $Recover$. An interpreter coupled with the execution monitor is defined by the relation $DoEM(\delta, s, s')$ similarly to the relation $Do(\delta, s, s')$ above (with *Trans** replaced by the reflexive transitive closure of $TransEM$). The most challenging part of this research endeavor is to find the declarative definitions of $Relevant$ and $Recover$ appropriate for a wide range of application domains. One possible class of monitors is considered in [2]: a discrepancy between mental model and the real world is deemed *relevant* if a remaining part of the Golog program cannot be successfully completed in the situation resulting after sensing; as a *recovery* technique it is suggested to generate a (possibly short) plan such that it may counter-balance a perceived discrepancy, then insert this plan as a prefix to the remaining part of program and continue the overall control process. Another recovery technique that I explore is to remove an unnecessary segment of a remaining part of the Golog program (this has to be done when an external process achieved already a result that the segment was designed to achieve). I'm also looking for definitions of $Relevant$ and $Recover$ appropriate to temporal, spatial domains and other cases. In the case of temporal Golog programs, if discrepancies are caused only by delays in time (the current time is greater than the scheduled time), then it seems appropiate to reschedule the remaing program (if possible). It remains to see whether suitably general definitions of $Relevant$ and $Recover$ can be formulated in other cases.

References

[1] G. De Giacomo, Y. Lespérance, and H.J. Levesque. Reasoning about concurrent executions, prioritized interrupts, and exogenous actions in the situation calculus. In *Proc. of the 15th IJCAI–97*, volume 2, pages 1221–1226, Nagoya, Japan, 1997.

[2] G. De Giacomo, R. Reiter, and M.E. Soutchanski. Execution monitoring of high-level robot programs. In *Principles of Knowledge Representation and Reasoning: Proc. of the 6th International Conference (KR'98)*, pages 453–464, Italy, 1998.

[3] T.L. Dean and M.P. Wellman. *Planning and control.* Morgan Kaufmann, San Mateo, Calif., 1991.

[4] T. Kelley. Modeling complex systems in the situation calculus: A case study using the dagstuhl steam boiler problem. In *Principles of Knowledge Representation and Reasoning: Proc. of the 5th International Conference (KR'96)*, Cambridge, Massachusetts, 1996.

[5] H.J. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science. Available at: http://www.ep.liu.se/ea/cis/1998/018/*, vol. 3, N 18, 1998.

[6] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. Golog : A logic programming language for dynamic domains. *J. of Logic Programming*, 31, N 1–3:59–83, 1997.

[7] S. McIlraith. Explanatory diagnosis: Conjecturing actions to explain obsevations. In *Principles of Knowledge Representation and Reasoning: Proc. of the 6th International Conference (KR'98)*, pages 167–177, Italy, 1998.

[8] J. Pinto. *Temporal Reasoning in the Situation Calculus, Ph.D. Thesis.* Dept. of Computer Science, Univ. of Toronto, 1994.

[9] R. Reiter. *KNOWLEDGE IN ACTION: Logical Foundations for Describing and Implementing Dynamical Systems.* A draft of the first eight chapters of a book. Available at `http://www.cs.toronto.edu/~cogrobo/`, 1998.

[10] E. Sandewall. Combining logic and differential equations for describing real-world systems. In *Principles of Knowledge Representation and Reasoning: Proc. of the 1st International Conference (KR'96)*, Toronto, Ontario, 1989.