Execution Monitoring of High-Level Robot Programs.

Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
degiacomo@dis.uniroma1.it

Ray Reiter
Department of Computer Science
University of Toronto
reiter@cs.toronto.edu

Mikhail Soutchanski
Department of Computer Science
University of Toronto
mes@cs.toronto.edu

Working Notes of the
4th Symposium on Logical Formalizations of Commonsense Reasoning
London, UK, Jan. 6 - Jan. 9, 1998.

The main question of this talk:

Is it true that a robot can be intelligent only if it was programmed by an intelligent and meticulous programmer ?

As a programming language we select a version of Golog which is a high-level language with operational semantics based on the situation calculus.

GOLOG = alGOL in lOGic

Motivation.

GOLOG programmer provides:

1. An incomplete description of an initial situation.
2. A set of successor-state axioms:
   - effects of actions that the robot may perform.
   - effects of children's actions and effects of changes that may occur in nature.
   - (may be also) actions of cats, dogs and other agents.
3. A set of precondition axioms: legality of robot's actions.
4. A non-deterministic program. The program prescribes a set of intended behaviors, one of which the robot is supposed to exhibit.

Our terminology:

Actions of agents other than robot + natural events = environmental (or exogenous) actions

Motivation (cont'd).

C'est la vie. In the real world:

- The initial situation may be different from what programmer has expected.
- Other agents and the nature may interfere in an unexpected way.
- No robot's actuators are perfect.
- Because of these and other problems, an execution of the program may get stuck and the robot will not achieve its goal.

Motivation (cont'd.)

YOUR CURRENT CHOICE:

your simple program	$+$	environmental actions	$\longmapsto$	$\dagger\ \dagger\ \dagger$ dead-end

OR

your complicated program	$+$	environmental actions	$\longmapsto$	$\surd$ success

The second alternative requires meticulous programming that takes into account the immense complexity of real-world environments.

WE OFFER:

your simple program	$+$	environmental actions	$+$	'predefined' execution monitor	$\longmapsto$	! ! $\heartsuit$!!

An execution monitor can be written once and forever: it will supervise operations of any Golog program.

Motivation (cont'd.)

Responsibilities of the execution monitor:

- sense environment
- determine whether an exogenous action was relevant or not
- if yes, call the recovery procedure
- the recovery procedure finds the shortest plan to counter-balance the exogenous disturbance

We specify an execution monitor in classical logic.

Benefits of logical approach:
We can

1. formulate general requirements which must be satisfied by any execution monitor
2. for each particular monitor verify whether it satisfies requirements.
3. assess the ranges of applicability of algorithms for execution monitoring developed so far.

All axioms and details of implementation are provided in the paper.

Golog.

Golog is a logic-programming language whose primitive actions are those of a background domain theory. It includes the following constructs:

nil,	empty program
a,	primitive action
$\phi?$,	test the truth of condition ϕ
$(\delta_1; \delta_2)$,	sequence
$(\delta_1 \mid \delta_2)$,	nondeterministic choice of two actions
$\pi v.\delta$,	nondeterministic choice of argument v to an action
δ^*,	nondeterministic iteration
$\mathbf{proc}\ P(\vec{v})\, \beta\ \mathbf{end}$,	procedure with formal parameters $\vec{v}$ and the body β.

Note that in contrast to straight line or partially ordered plans a Golog program can be arbitrary complex, i.e. it may include `while`-loops, recursive procedures and nondeterministic choices.

There are several interpreters for Golog programs:

- off-line
- on-line: 1). cautious, 2). brave

All interpreters are implemented in Prolog.

Programmer's responsibility.

For each application domain a programmer must provide:

1. The specification of all primitive actions (robot's and exogenous) and their effects, together with an axiomatization of the initial situation.
2. A Golog program that may or may not take into account exogenous actions occurring when the robot executes the program.

Examples of the specification and the Golog program are given on the following 2 slides.

The specification of all actions.

In our blocks–world example all primitive actions are specified by the following set of axioms.

`Successor State Axioms.`

$$on(X,Y,do(A,S)) \equiv A = move(X,Y) \vee \\ on(X,Y,S) \wedge A \neq moveToTable(X) \wedge A \neq move(X,Z).$$

$$ontable(X,do(A,S)) \equiv A = moveToTable(X) \vee ontable(X,S) \wedge A \neq move(X,Y).$$

$$clear(X,do(A,S)) \equiv (\exists Y,Z).(A = move(Y,Z) \vee \\ A = moveToTable(Y)) \wedge on(Y,X,S) \\ \vee clear(X,S) \wedge A \neq move(Y,X).$$

`Action Precondition Axioms.`

$$poss(move(X,Y),S) \equiv clear(X,S) \wedge clear(Y,S) \wedge X \neq Y.$$

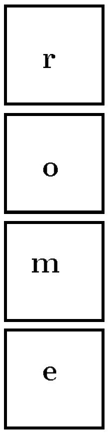
$$poss(moveToTable(X),S) \equiv clear(X,S) \wedge \neg ontable(X,S).$$

The implementation of these specifications in Prolog is straightforward.

Blocks–world program.

The following is a blocks world Golog program that nondeterministically builds a tower of blocks spelling “paris” or “rome”.

$$\textbf{proc}\ tower \quad (\ makeParis \mid makeRome\ ) \quad \textbf{endProc}.$$



$\textbf{proc}\ makeRome$
$\quad \pi\, b_0.[e(b_0) \wedge ontable(b_0) \wedge clear(b_0)]?\,;$
$\qquad \pi\, b_1.m(b_1)?\,; move(b_1, b_0)\,;$
$\qquad\quad \pi\, b_2.o(b_2)?\,; move(b_2, b_1)\,;$
$\qquad\qquad \pi\, b_3.r(b_3)?\,; move(b_3, b_2)$
$\textbf{endProc}$

$\textbf{proc}\ makeParis$
$\quad \pi\, b_0.[s(b_0) \wedge ontable(b_0) \wedge clear(b_0)]?\,;$
$\qquad \pi\, b_1.i(b_1)?\,; move(b_1, b_0)\,;$
$\qquad\quad \pi\, b_2.r(b_2)?\,; move(b_2, b_1)\,;$
$\qquad\qquad \pi\, b_3.a(b_3)?\,; move(b_3, b_2)$
$\qquad\qquad\quad \pi\, b_4.p(b_4)?\,; move(b_4, b_3)$
$\textbf{endProc}$

Note that this program does not take into account exogenous actions that may occur when the robot executes the program. If execution of the program would not be monitored, external disturbances could jeopardize the execution of the program.

Blocks-world example.

A programmer also provides a first order sentence describing the program's goal, or what programmers call a program *postcondition*.

$$goal(s) \equiv spellsParis(s) \vee spellsRome(s),$$

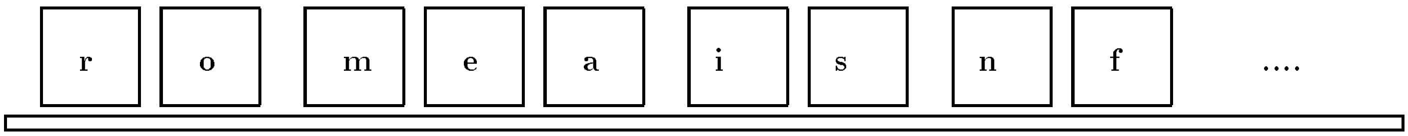
$$\begin{aligned} spellsRome(s) \equiv (\exists b_0, b_1, b_2, b_3).&e(b_0) \wedge ontable(b_0, s) \wedge \\ &m(b_1) \wedge on(b_1, b_0, s) \wedge \\ &o(b_2) \wedge on(b_2, b_1, s) \wedge \\ &r(b_3) \wedge on(b_3, b_2, s) \wedge clear(b_3, s). \end{aligned}$$

The $spellsParis(s)$ predicate is defined similarly.

This postcondition is appended to the program. In other words, we shall be dealing in this paper with the program

$$tower\,;goal?$$

In our scenario, the initial situation is the following:



Note that there is no block with the letter "p". We selected this arrangement intentionally to illustrate the difference between cautious and brave interpreters.

Execution monitoring.

The on-line interpreter executes a nondeterministic program step-by-step. When it makes a nondeterministic choice, it looks ahead and selects that branch which leads towards the successful completion of the program.

Important note: *robots cannot backtrack, because they do their actions in the real world.*

Whenever a primitive action has to be performed or a test has to be evaluated, the monitor gets new sensory input from the environment and if necessary recovers from unanticipated exogenous disturbances.

This cycle of on-line interpreting, sensing and recovering (if necessary) repeats until the program terminates.

Simplifying assumption: a sensory input is given to the monitor in terms of an exogenous action (or external program). In the implementation, a user simulates environmental actions.

An execution trace: Step 1.

Because in our example there is no "p" block, the procedure $makeParis$ cannot be successfully completed.

Hence, a cautious on-line interpreter selects procedure $makeRome$.

It does the first step of the execution:

$$\pi\, b_0.[e(b_0) \wedge ontable(b_0) \wedge clear(b_0)]?\,;$$

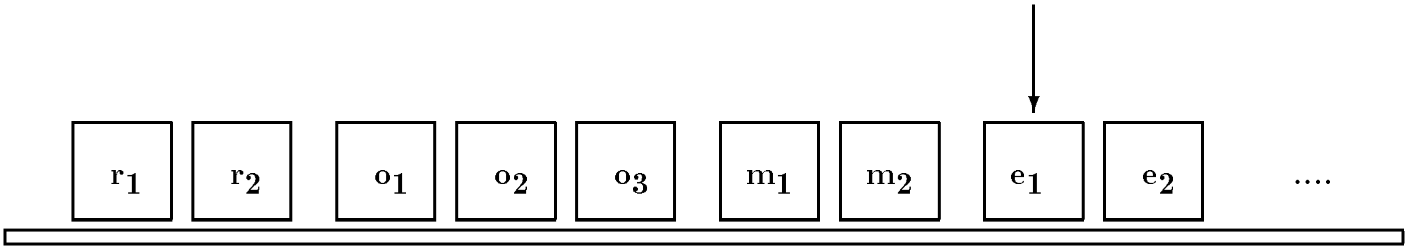
by determining that the block e_1 will be the basis of a goal tower.

The remainder of the program is:

nil ;
$\pi\, b_1.m(b_1)?\,; move(b_1, e1)\,;$
$\quad \pi\, b_2.o(b_2)?\,; move(b_2, b_1)\,;$
$\quad\quad \pi\, b_3.r(b_3)?\,; move(b_3, b_2)\,;$
$(goal)?$

where nil results after the first step.

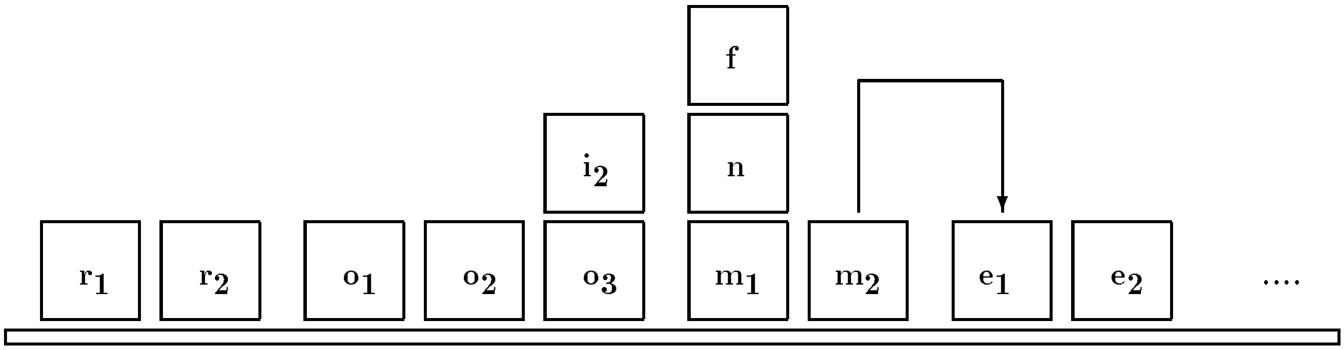
Part of the initial situation (blocks with other letters are not shown).



An execution trace: the 1st exogenous disturbance.

An external agent does: move(n,m1) ; move(f,n) ; move(i2,o3)

The resulting situation is: do(move(i2,o3),do(move(f,n),do(move(n,m1),s0)))



Because the remaining program

nil ;
$\pi\, b_1.m(b_1)?\,; move(b_1, e1)\,;$ $\surd$
$\quad \pi\, b_2.o(b_2)?\,; move(b_2, b_1)\,;$
$\quad\quad \pi\, b_3.r(b_3)?\,; move(b_3, b_2)\,;$
$?(goal)$

still can be successfully completed (pick m_2, move it on top of e_1, etc.)
$\Longrightarrow$

$\boxed{\textbf{no recovery is necessary}}$.

Execution trace: 2nd step & 2nd exogenous disturbance.

Remainder of the program:

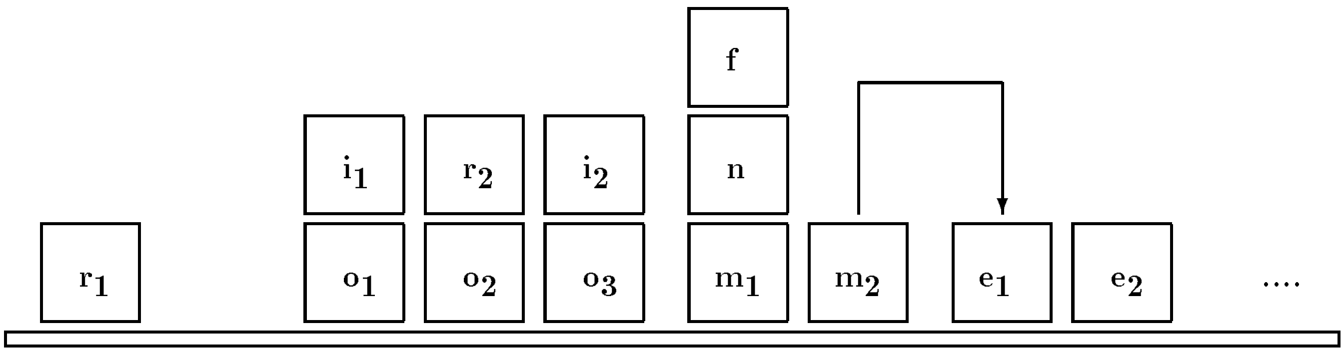
nil ;
  $move(m2, e1)$;
    $\pi\, b_2.o(b_2)?\, ; move(b_2, m2)$;
      $\pi\, b_3.r(b_3)?\, ; move(b_3, b_2)$;
$(goal)?$

Current situation: do(move(i2,o3),do(move(f,n),do(move(n,m1),s0)))

An external agent does: move(i1,o1) ; move(r2,o2)

The resulting situation is:

do(move(r2,o2),do(move(i1,o1),do(move(i2,o3),
do(move(f,n),do(move(n,m1),s0)))))



An execution trace: recovery procedure

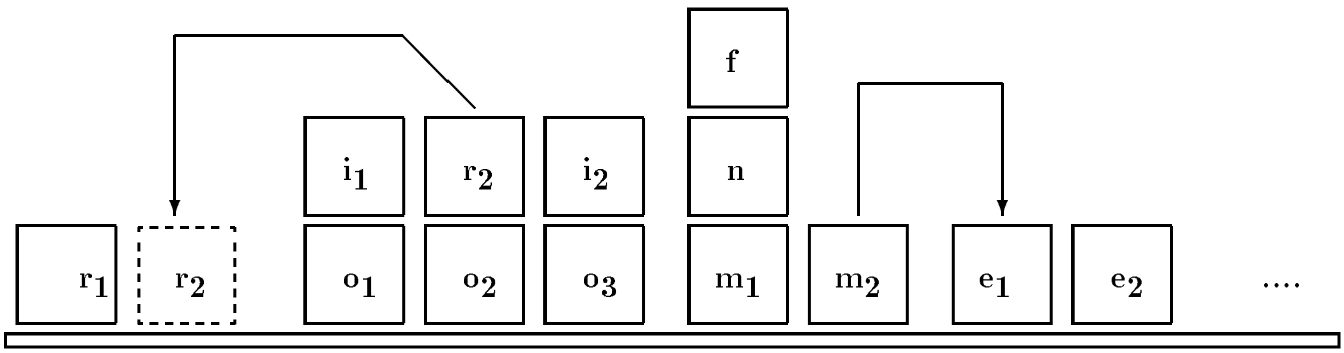
The remaining program cannot be completed $\Longrightarrow$
the monitor calls the recovery procedure.

$\boxed{\text{We consider a simple kind of } \textbf{planner} \text{ as the recovery procedure.}}$.

The recovery procedure finds the primitive action

$$moveToTable(r2)$$

and determines that executing it followed by the rest of the program results in successful termination.



etc.
The rest of the program can be traced similarly.

Conclusions.

Is it true that a robot can be intelligent only if it was programmed by an intelligent and meticulous programmer ?

Answer:

A simple program
which does not anticipate all environmental actions
can still produce intelligent behavior
by using an intelligent execution monitor.

Conclusions (cont'd).

Plans for ongoing and future work include the following issues:

1. Draw closer parallels with the concept of controllable systems in discrete event control theory.
2. Prove soundness and/or completeness of various recovery procedures.
3. Extend these ideas to temporal domains, for example, monitoring robot control programs written in temporal Golog.
4. Implement these ideas on the Cognitive Robotics Group's RWI B21 autonomous robot at the University of Toronto.