GOLOG: an Elevator Controller

Prof. Mauro Gaspari
Dipartimento di Informatica Scienze e Ingegneria (DISI)
mauro.gaspari@unibo.it

GOLOG

- GOLOG allows to express complex action expressions. It can be viewed as a programming language whose semantics is defined via macro-expansion into sentences of the situation calculus.
- We call this language GOLOG, for “alGO in LOGic.” GOLOG appears to offer significant advantages over current tools for applications in dynamic domains like the high-level programming of robots and software agents, process control, discrete event simulation, etc.
- Here we present a simple example.

Procedures

- The GOLOG approach is to define complex action expressions using some additional extralogical symbols (e.g., while, if, etc.) which act as abbreviations for logical expressions in the language of the situation calculus.
- These extralogical expressions should be thought of as macros which expand into genuine formulas of the situation calculus.

The Do Predicate

- The do predicate here takes 3 arguments: a GOLOG action expression, and terms standing for the initial and final situations.
- Normally, a query will be of the form do(e, SO, S), so that an answer will be a binding for the final situation S.
- In this implementation, a legal GOLOG action expression used for the definition of procedures include the expressiond defined in the next slide.

Legal GOLOG Actions for procedures

- $[e_1, \ldots, e_n]$, sequence.
- $?(p)$, where p is a condition
- $e_1 \# e_2$, nondeterministic choice of e_1 or e_2.
- $\texttt{if}(p, e_1, e_2)$, conditional.
- $\texttt{star}(e)$, nondeterministic repetition.
- $\texttt{while}(p, e)$, iteration.
- $\texttt{pi}(v, e)$ nondeterministic assignment, where v is an atom (standing for a GOLOG variable) and e is a GOLOG action expression that uses v.
- a, where a is the name of a user-declared primitive action or defined procedure

Actions and Fluents

- **Actions**:
  - up(n): Move the elevator up to floor n.
  - down(n): Move the elevator down to floor n.
  - tumofl(n): Turn off call button n.
  - Open: Open the elevator door.
  - Close: Close the elevator door.
- **Fluents**:
  - current_floor(s)= n: In situation S, the elevator is at floor n.
  - on(n, s): In situation s call button n is on.
  - next_fIoor(n,s): In situation S the next floor to be served is n.

Actions Preconditions

$$Poss(up(n), s) \equiv current\_floor(s) < n.$$
$$Poss(down(n), s) \equiv current\_floor(s) > n.$$
$$Poss(open, s) \equiv true.$$
$$Poss(close, s) \equiv true.$$
$$Poss(turnoff(n), s) \equiv on(n, s).$$

Successor State Axioms

- It uses $\supset$ in place of $\rightarrow$ and $\equiv$ in places of $\leftrightarrow$

$$Poss(a,s) \supset [current\_floor(do(a,s)) = m \equiv \{a = up(m) \vee a = down(m) \vee current\_loor(s) = m \wedge \neg(\exists n)a = up(n) \wedge \neg(\exists n)a = down(n)\}].$$

$$Poss(a,s) \supset [on(m, do(a,s)) \equiv on(m,s) \wedge a \neq turnoff(m)].$$

Fluents

- This fluent defines the next floor to be served as a nearest floor to the one where the elevator happens to be:

$$next\_floor(n,s) \equiv on(n,s)$$
$$\wedge(\forall m).on(m,s) \supset |m - current\_floor(s)| \geq |n - current\_floor(s)|.$$

The Golog Procedures

proc *serve*(n) *go_floor*(n); *turnoff*(n); *open*; *close* **endProc**.

proc *go_floor*(n) $(current\_floor = n)?|up(n)|down(n)$ **endProc**.

proc *serve_a_floor* $(\pi\, n)[next\_floor(n)?;\ serve\ (n)]$ **endProc**.

proc *control* [**while** $(\exists n)on(n)$ **do** *serve_a_floor* **endWhile**]; *park* **endProc**.

proc *park* **if** $current\_floor = 0$ **then** *open* **else** $down(0)$; *open* **endIf endProc**.

Prolog Implementation

- **Primitive Control Actions**:

```
primitive_action(turnoff(N)).  % Turn off call button N.
primitive_action(open).        % Open elevator door
primitive_action(close).       % Close elevator door.
primitive_action(up(N)).       % Move elevator up to floor N.
primitive_action(down(N)).     % Move elevator down to floor N.
```

Procedures

proc(goFloor(N), ?(currentFloor(N)) # up(N) # down(N)).

proc(serve(N), goFloor(N) : turnoff(N) : open : close).

proc(serveAfloor, pi(n, ?(nextFloor(n)) : serve(n))).

proc(park, if(currentFloor(0), open, down(0) : open)).

/* control is the main loop. So long as there is an active call button, it serves one floor. When all buttons are off, it parks the elevator. */

proc(control, while(some(n, on(n)), serveAfloor) : park).

Preconditions for Primitive Actions

poss(up(N),S) :- currentFloor(M,S), M < N.

poss(down(N),S) :- currentFloor(M,S), M > N.

poss(open,S).

poss(close,S).

poss(turnoff(N),S) :- on(N,S).

Successor State Axioms for Primitive Fluents

currentFloor(M,do(A,S)) :- A = up(M) ; A = down(M) ;

not A = up(N), not A = down(N), currentFloor(M,S).

on(M,do(A,S)) :- on(M,S), not A = turnoff(M).

Initial Situation

% Call buttons: 3 and 5. The elevator is at floor 4.

on(3,s0).
on(5,s0).
currentFloor(4,s0).

- This last axioms specifies that, initially, buttons 3 and 5 are on, and moreover no other buttons are on. In other words, close world assumption is assumed. This is essential for the Prolog version.

/* nextFloor(N,S) is an abbreviation that determines which of the active call buttons should be served next. Here, we simply choose an arbitrary active call button. */

nextFloor(N,S) :- on(N,S).

Running the elavator

```
?- do(pi(n,[?(on(n)),turnoff(n)]),s0,S).

S = do(turnoff(3),s0) ;

S = do(turnoff(5),s0) ;

no

- ----------------------------------------

?- do(pi(n, turnoff(n) # ([?(next_floor(n)),go_floor(n)])),s0,S).

S = do(turnoff(3),s0) ;

S = do(turnoff(5),s0) ;

S = do(down(3),s0) ;

S = do(up(5),s0) ;

no
```

We ask the interpreter to pick a floor and turn off its call button.

We ask the interpreter to either turn off a call button or to go to a floor that satisfies the test next-floor.

Example

```
?- do(control,s0,S).

S = do(open,do(down(0),do(close,do(open,do(turnoff(5),do(up(5),do(close,
do(open,do(turnoff(3),do(down(3),s0)))))))))) ;

S = do(open,do(down(0),do(close,do(open,do(turnoff(3),do(down(3),do(close,
do(open,do(turnoff(5),do(up(5),s0)))))))))) ;

S = do(open,do(down(0),do(close,do(open,do(turnoff(5),do(up(5),do(close,
do(open,do(turnoff(3),do(down(3),s0)))))))))) ;

S = do(open,do(down(0),do(close,do(open,do(turnoff(3),do(down(3),do(close,
do(open,do(turnoff(5),do(up(5),s0)))))))))) ;

no
```

In the final query, we call the main elevator controller, **control**, to serve all floors and then park the elevator.