IndiGolog vs. Golog

Golog Interpreter

- File named gologinterpreter.pl
- Fluents are boolean
- Do not require the declaration of fluents (we can omit *prim_fluent(.)*)
- Require *restoreSitArg* atoms: allows 'situation calculus' style encoding of domains

Indigolog Interpreter

- In the file named golog.pl
  - NOTE: this file should not be changed unless you are you know what you are doing
- Deal with *multivalued fluents*
- Require specification of fluents by *prim_fluent*
- Do not require atoms of the form *restoreSitArgs*
- Provide rules to deal with a high-level description of action domains
- Run a procedure with the command
  indigolog(procedure_name)

Action Domains in IndiGolog

- Fluents
  - multivalued
  - declared by *prim_fluent*(.)
- Actions
  - declared by *prim_action*(.)
  - executability condition specified by
    poss(Action,Condition)
  - Effects specified by
    causes(Act,Fluent,Value,Condition)
- Initial state
  - Specified by atoms *initially(Fluent, Value)*

The Elevator Domain in IndiGolog

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

% Primitive fluent             }
                               }
prim_fluent(currentFloor).     }
prim_fluent(opened).           }   NOT NEEDED in GOLOG
prim_fluent(on(N)):- floor(N). }
                               }
```

Executability Condition

% Preconditions for Primitive Actions.

poss(up(N), and(currentFloor = M, M < N)).

poss(down(N), and(currentFloor = M, $\underbrace{\text{M > N}}$)).

poss(open, true).

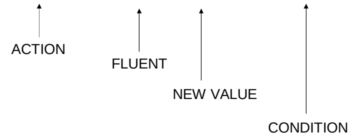
poss(close, true).

poss(turnoff(N), on(N) = true).

IT WAS DIFFERENT IN GOLOG

Effects of Actions

```
causes_val(down(N), currentFloor, N, currentFloor >N).
causes_val(up(N), currentFloor, N, currentFloor < N).
causes_val(open, opened, true, true).
causes_val(close, opened, false, true).
causes_val(turnoff(N), on(N), false, currentFloor = N).
```



ACTION

FLUENT

NEW VALUE

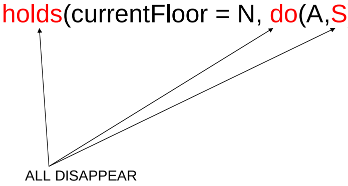
CONDITION

Initial State

```
initially(on(3), true).
initially(on(5), true).
initially(currentFloor, 4).
```

What are not needed?

```
holds(currentFloor = N, do(A,S)):- ….
```



ALL DISAPPEAR

Procedures

```
% Definitions of Complex Control Actions

proc(goFloor(N), [ndet(?(currentFloor = N), ndet(up(N), down(N)))]).
proc(serve(N), [goFloor(N), turnoff(N), open,  close]).
proc(serveAfloor, pi(n, [?(on(n) = true), serve(n)])).
proc(park, if(currentFloor = 0, open, [down(0), open])).

proc(control, [while(some(n, on(n) = true), serveAfloor),park]).
```

Almost the same as in GOLOG with the following differences:

- '#' is replaced with *ndet* and is written in prefix notation:
  p1#p2 is written as ndet(p1,p2)
- ':' is replaced by the list notation of PROLOG
  p1:p2 is written as [p1, p2]
- Expressions are used in place of testing condition

Preparation for connecting to RCX

```
% actionNum(?Act, ?ActNumber): Returns ActionNumber associated
% with Action and vice versa.

actionNum(open, 0).
actionNum(close, 1).
actionNum(turnoff(N), N1):- floor(N), N1 is N+2.
actionNum(down(N), N2):- floor(N), N2 is N+8.
actionNum(up(N), N3):- floor(N), N3 is N+14.
```

} THIS IS NEW

Each action should be associated with a behavior implemented in the corresponding LEGO program. This number is sent to the RCX for execution of the action. RCX also returns a number reporting the occurrence of exogenous Action

elevator.pl

- Contains the description and procedures
- Can be used to test the action theory by
  - compiling in Eclipse
  - testing with the command *indigolog(control)*
- This program **does not connect to RCX** as the program delivery.pl. It has a set of rules that simulates the connection to RCX (this rules come after the atoms defining the action numbers)

Connecting to RCX

- File ‘main.pl’
- Only needs to change the *main* predicate.

Some ideas for the project

- Create a theory (fluents, actions, …) for the problem
- Test it without connecting to the RCX (make sure that the procedures are working)
- Test the basic behaviors of the robot work fine
- Combine them and worry about the exogenous actions