Rule Control of Teleo-Reactive, Multi-tasking, Communicating Robotic Agents

Robotic agent programming in QuLog and TeleoR

Keith Clark
Imperial College
University of Queensland
University of New South Wales

Joint work with
Peter Robinson
University of Queensland

Baxter two arm concurrent tower building
QuLog - a modern logic programming language

- Flexibly typed, multi-threaded, higher order

- Relation and function defining rules - The declarative subset
  - Relation rules more declarative than Prolog
  - Relations must have their modes of use declared
    - which arguments must be given, which may be returned
  - Used to encode the agent’s knowledge

- Dynamic relations - defined only by facts
  - Like tuples of a relational DB
  - Used for the agent’s dynamic beliefs – its Belief Store (BS)

- Top layer of action rules – the imperative subset
  - Threads execute actions
  - Primitive actions – thread forking, I/O, inter-agent comms, BS updating

TeleoR

- Application specific extension of QuLog to facilitate programming of robust, goal directed, concurrently executing robotic device control threads

- Major development of Nilsson’s T-R robotic language of guard -> action rules sequenced into parameterised procedures

- T-R has its roots in the triangular table representation of the generalised plans of first AI robot – SRI’s Shakey of late 1960s

- TeleoR’s rule guards are QuLog queries to the agent’s BS, optionally using its knowledge

- Compile time guarantee of fully determined and type correct robot actions

- High level multi-tasking using task atomic procedures

- Formal state transition semantics
Minimal Multi-threaded Agent Architecture

- **Declarative QuLog**
- **Imperative QuLog**
- **Percepts Handler**
- **Message Handler**
- **Task Thread**
- **TeleoR + imperative QuLog**
- **Pub/Sub and Addressed Message Router**
- **Symbolic messages**
- **Fixed Knowledge - Relation and Function definitions**
- **Belief Store - updateable facts**

Extra threads for extra capabilities

- **Fixed Knowledge Rules**
- **Updateable Beliefs**
- **Common Ontology for all threads**
- **Percepts Handler**
- **Message Handler**
- **Task Thread**
- **Temporary Query Thread**
- **SLAM Thread**
- **CHR Thread**
- **Abducting new beliefs. resolving inconsistency**
One arm block tower building

Task Knowledge

```
def block::= 1..9

durative pickup(block), put_on_block(block), put_on_table()
percept on(block,block), on_table(block), holding(block)

rel sub_tower(list(block)), tower(list(block)), clear(block)
fun top(list(block)) -> block, tail(list(block)) -> list(block)

sub_tower([B]) <= on_table(B)
sub_tower([B1,B2,..Blocks]) <=
  on(B1,B2) & sub_tower([B2,..Blocks])
tower([B,..Bs]) <= clear(B) & sub_tower([B,..Blocks])
clear(B) <= not exists OtherB on(OtherB,B)
top([B,.._]) -> B
tail([_,..Bs]) -> Bs
```
Control Knowledge

Universal conditional top-level plan for block tower building

tel makeTower(list(block))

makeTower(Blocks){

tower(Blocks) -> () % Goal holds, do nothing
sub_tower(Blocks) -> make_clear(top(Blocks))
tower(tail(Blocks)) -> move_to_block(head(Blocks), head(tail(Blocks)))
Blocks=[B] -> move_to_table(B)
true -> makeTower(tail(Blocks))
}

Mutually recursive control procedures

tel make_clear(block)
make_clear(Block){
clear(Block) -> () % Goal is achieved
on(OnBlock,Block) -> move_to_table(OnBlock)
}
tel move_to_table(block)
move_to_table(OnBlock){
on_table(OnBlock) -> () % Goal is achieved
holding(OnBlock) -> put_on_table()
clear(OnBlock) & not holding(_) -> pickup(OnBlock)
not holding(_) -> make_clear(OnBlock)
holding(_) -> put_on_table()
}
Concurrent Task Tower Building

Update of single task tower builder

Just need to add:

\texttt{task\_start makeTower}

\texttt{task\_atomic move\_to\_table, move\_to\_block}
Multi-tasking architecture

Co-operative navigation

Blue robot path [(blue, 1, 0), (green, 5, 1), (yellow, 4, 5), (red, 8, 4)]
Common beliefs of agents

- The topological map of rooms and doors. e.g. connected(blue,1,2,90), connected(blue,2,1,270)
- Which chargers are 'reserved'
- Which doors are closed
  - this may be inaccurate
- Location of each robot
- Each robot's current path, if any

The task knowledge

def room ::= 0..15
def door_status ::= open | shut
def battery_status ::= high | low
def door ::= blue | green | yellow | red
def path == list((door, room, room))
% type macro
def message ::= new_loc(robot, room) | path(robot, path) |
  staying(robot) | backed_up(robot, room, door) | ..
% message is a reserved type name
durative turn(turn_dir), forward(), reverse()
percept battery(battery_status), see_door(door, door_status),
  see_centre_ahead(), see_centre_close(), at_room_centre(),
  in_doorway(door), facing(door)

dyn loc(robot, room), open(door, room), reserved(robot, room),
  following(robot, room, path)

rel connected(?door, ?room, ?room), charger_room(?room),
  home_room(?robot, ?room), ...
Recursive path follow procedure

tel follow(robot, room, path)

follow(Me, DestRm, Path) {

loc(Me, DestRm) ~> ()

loc(Me, Rm) & Path=[ (Door, Rm, DestRm), ... ] &
    connected(Door, Rm, DestRm, DoorDir) ~> 
    move_to_next_room(Me, Rm, DestRm, DoorDir)

Path= [ (_, Rm, DestRm), ... PriorPath ] ~> 
    follow(Me, Rm, PriorPath)
%
recursive call to get robot into Rm
}

Top level navigation

tel get_to_room(robot, rel(?room)) % Higher order, 2nd arg a monadic rel.
get_to_room(Me, DestTest) {

loc(Me, MyRm) & DestTest(MyRm) ~> ()

following(Me, DestRm, Path) &
    rem_path_open_and_shortest_to_dest(Me, Path, DestTest) ~> 
    follow(Me, DestRm, Path)

loc(Me, MyRm) &
    graph_path(MyRm, DestRm, Path, open_connected, DestTest)) ~> 
    () ++ new_path(Me, DestRm, Path) to pedro

true ~> stay_avoiding_other_robots(Me) ++ staying(Me) to pedro
}

rel open_connected(door, room, room)
open_connected(Door, Rm1, Rm2) <= 
    connected(Door, Rm1, Rm2, _) & open(Rm1, Door)
Minimal Multi-threaded Agent Architecture

**Fixed Knowledge - Relation and Function definitions**

**Belief Store - updateable facts**

- Percept facts
- Message Handler: ag@host
- Atomic Updates

Frequent reconsideration of fired rules

Atomic Updates

Atomic Updates

Mid-level action control messages

Outgoing new_path message.

Incoming new_path messages - serialised

Delivery robots in Japan

https://youtu.be/_QndP_PCpSw
TeleoR semantics and implementation

- Formal State Transition Semantics
- Optimized Reference Implementation
  - Runtime system that implements state transition semantics
  - Compile time analysis ensures rule guards are not re-evaluated on BS update if no relevant change made
- Currently compiled to multi-threaded Qu-Prolog
- Will be compiled to specialized Abstract Machine Code similar to but simpler than Warren's Prolog Abstract Machine Code

Sources and software

Clark & Robinson, Robotic Agent Programming in TeleoR, ICRA 2015, IEEE


Clark et al, A Framework for Integrating Symbolic and Sub-symbolic Representations, IJCAI 2016, AAAI Press


first 5 chapters at teleoreactiveprograms.net

TeleoR and QuLog Software for Unix, Linux and OS X at http://staff.itee.uq.edu.au/pjr/HomePages/QulogHome.html

Collaboration and users welcomed