LEGOLOG

an implementation of Golog for controlling LEGO® MINDSTORMS™

Robots

http://www.cs.toronto.edu/cogrobo/

http://mag.usr.dsi.unimi.it/

Cognitive robotics: the big picture

- We write a high-level description of robot’s action capabilities, in the language GOLOG
- Prolog interpretation of our GOLOG theory generates apt calls to (NQC) execution routines and to two-ways communication with the robot.
- Such theory is integrated with low-level routines (partly in NQC) for sensing, acting and reacting

The big picture, cont’d

- [re]-Planning is in GOLOG and computer-side
- Sensing and acting is in NQC and robot-side
- Communication is asynchronous

(this is the technically most challenging issue)

in standard MINDSTORMS,

- the robot executes an absolute plan, sent in by the computer
- no action failure analysis, no re-planning
- no sensing to drive the plan (only execution)

Program issues

1. Legolog
2. Golog programming language
3. NQC and Legolog
4. Case study: The Delivery Robot
5. Besides Golog

1. Legolog

- Lego Mindstorms (from MIT's Intelligent Brick)
- Legolog Idea
- Legolog schema
- Comunication Protocol

Lego Mindstorms RIS

RCX (Robotic Command Explorer)

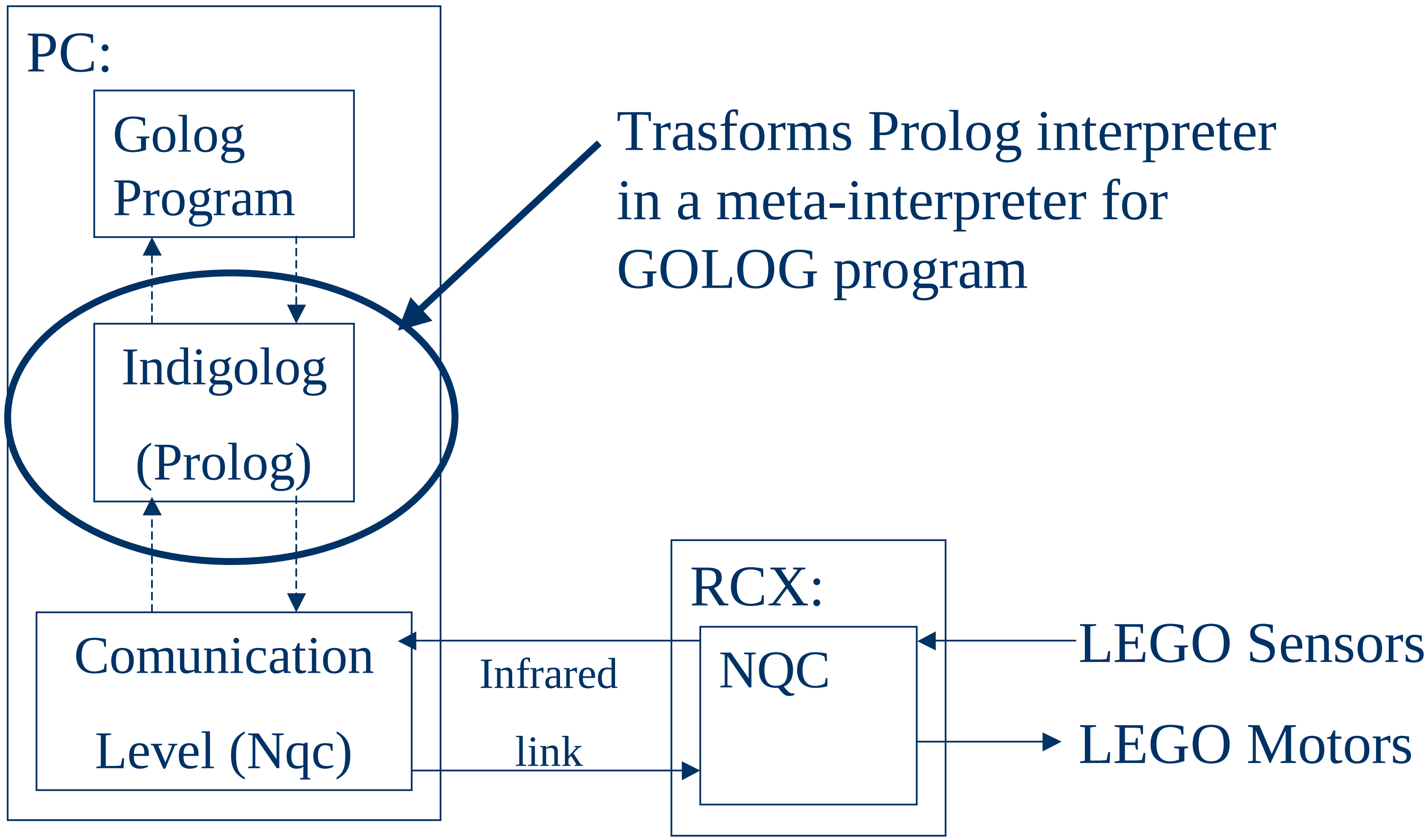
- Hitachi H8/3297 microprocessor
- 3 inputs
  - Pushbutton, light, temperature, rotation
- 3 outputs
  - Motors, light
- Infrared comunication port → tower →pc serial port
- Programming: LEGO®, NQC, LegOs and more

Idea: write control program on standalone computer and dowload to RCX

Legolog: basic idea

- Primitive actions are in RCX (simple behaviour)
- Languages used: Indigolog [interpreted in] Prolog and NQC
- Comunication is done via infrared tower
- For technical reasons, Prolog initiates all comunication
  - Golog determines next action and sends message to RCX, which must acknowledge within 3.5 seconds with sensing value
  - Golog can “query” RCX to know if exogeneous actions occured
- Indigolog interpreter treats concurrency, interrupts and exogeneous actions

Legolog schema



Reasoning

- RCX does no reasoning
- Golog decides what primitive actions to perform and sends action codes to RCX
- Golog monitors exogenous actions and sensing information from the RCX
- The Golog interpreter runs on top of Prolog on a standalone pc, equipped with a IR tower

Legolog comunication protocol

- Desiderable: send/receive arbitrarily large (>0) numbers
  - Multiple RCXs
  - Arbitrary sensing values
- How to
  - Send numbers 1 <= n <= 7 bits at a time
  - Use “continuation bit”
  - Handful of special messages
- Prolog initiates all comunication
  - RCX would wait for Golog anyway

2. Golog: Logic programming Language for Dynamic Domains

- General features
- Situation Calculus
- Domain representation

Golog features

- Explicit representation of the dynamic world being modeled
- Based on logic of actions (situation calculus):
  - Preconditions – action – Effects
- High level of abstraction
- Run-time queried interpreter
- Handles concurrency (Indigolog)

… GOLOG is very rich, we use only fragments

Golog features (2)

- GOLOG : alGOL in LOGic
- Supports: sequence, conditionals, loops, non-deterministic choice; concurrency, priorities, interrupts, exogenous actions, sensing
- Primitive statements: domain-dependent actions to be executed by the agent
- Conditions/tests: domain-dependent predicates(fluents) affected by actions
- Action theory: precondition axioms, successor state axioms
- Find sequence of actions that constitutes legal execution of high-level program

Situation Calculus

- Situation = *state* (more precisely, a history of D)
- State is referred to as:
  - *init* : initial state
  - *do(A,S)* : state resulting from doing action A in S
- we focus on situations that can be achieved: predicate *poss(A,S)* caractherizes when action A is executable in state S

Example situation

init

do(move(rob,s109,s103), init)

do(move(rob,s103,mail),
    do(move(rob,s109,s103),
        init))

do(pickup(rob,k1),
    do(move(rob,s103,mail),
        do(move(rob,s109,s103),
            init)))

Representing a Domain

- A domain of application is specified by the union of the following sets of axioms:
  - *init* – what is true in the initial state:
    - holds(at(robot,s109), init)
  - *fluent(name)* – representing boolean entities:
    - fluent(location)
  - *primitive (atemporal) relations* – unique names axioms
  - *poss(A,S)* – Action Precondition axioms, one for each primitive action
  - *do(A,S)* – successor state axioms, one for each fluent

3. NQC and Legolog

- Nqc code
- Example: main loop

NQC for Legolog

- *Not Quite C* (NQC) is an independent C-like programming language
- Used to realise firmware-virtual machine
- NQC programs get dowloaded on RCX via infrared tower
- Comunication level

NQC primitives for Legolog

- ***initialize***: initalizes RCX, start exogenous action monitors, etc.
- ***startBehaviour***: determines which behaviour to perform on input
- ***panicAction***: what to do when Prolog not responding to RCX
- Additional code for behaviours, exogenous event monitoring, functions, etc.

nqc main loop

```
initialize();
while (true) {
    if (status == ABORT) {stopAllBehaviours(); status = OK; }
    if (status == PANIC) {panicAction();                          //beep, move around, etc.
                          SendMsg(PANIC_MSG);
                          ReceiveMsg(result); }                   //Hope for an abort command
    if (status == OK) { ReceiveMsg(result);
                        if (validActionMsg(result)) {
                            startBehaviour(result);
                            SendMsg(sensingValue); }              //Return sensor value
                        else if (exogRequestMsg(result)) {
                            SendMsg(exogAction);
                            exogAction = NO_EXOG_ACTION; } }
}
```

4. Case study: The Delivery Robot

- Scenario
- Golog Delivery Task
- Legolog files

Scenario

- Robot's world is a black-tape track, interrupted by *stations*, in bright color (other solutions are possible)
- Behaviour: pick up a package from one station and deliver it to another station
- Single-line road:
  - Turnaround to go backward
  - Numbered stations (1..6)
- When there are no more deliveries pending, robot returns to its initial state.

Pagnucco vs. AI-MI'01 class delivery robot

- On arriving in a From station, the robot waits a "continue" command.
- if the robot hits an unidentified objects, then all behaviours are stopped
- start position = 3

- The robot detects if all is in the right place by the sense buttons.
- if the robot hits an obstacle, then it moves it off track and continues
- start position = 0

Delivery commands

At run-time, we may give *exogenous* requests via an interaction window run by the Prolog:

- Delivery request: +(From, To).(*)
- Cancellation request: -(From, To).
- Delivery requests may be received at any time
- Cancellation requests must be made before the robot has collected the object from the "From" station.

(*) final period is important since the input must be in the form of a Prolog term

The Legolog files

- main_XXX.pl
- golog.pl (*)
- delivery.pl
- legorcx.pl (*)
- lego_XXX.pl (*)
- control.nqh (*)
- delivery.nqc
- delivery.nqh

XXX::=swi | ecl | lpa

(*) application-independent

main_XXX.lp

- short Prolog program that loads the rest of the Prolog files, as well as the indigolog interpreter (main control procedure)
- defines special implementation dependend predicates
- deals with exogeneous events that do not originate from the RCX

golog.pl(*)

- Defines the golog (IndiGolog) language
- before and after running a program it does any application dependent initialization and cleanup:

  initialize, …, finalize.

  - do the action, return the sensing result:

  execute(action, history, result)

- performs rolling forward to bound the length of the history of actions (Mainteinance action: rolling_down_the_river)
- check if anything has happened exogenously since the last time and return a list of actions, by repeatedly calling:

  exog_occurs(list-of-actions)

delivery.pl

It is the application program written in Golog.

1. Declarative part: specify all axioms for an application-dependent action theory (fluent, primitive and exogenous actions…)

2. Procedural part: defines a top level program called "control" that is a set of prioritized interrupts

3. Interface Golog-RCX: initialization procedures and message sending/receiving defined in legorcx.pl.

legorcx.pl(*)

High-level routines for comunication between the interpreter and the LEGO RCX, in Prolog.

The main predicate defined are:

- sendRcxActionNumber(number, result)
- receiveRcxActionNumber(list-of-numbers)

called in delivery.pl and returning a sensing value or a list of number for actions to be executed.

lego_XXX.pl(*)

This file defines lowest level communication and timing predicates for the various Prolog implementations:

- Open serial port for readint/writing
- Read/write a byte from/to the RCX
- Close the serial port

This predicates are only called from within legorcx.pl.

control.nqh(*)

Application independent part of the NQC code.

It contains:

- routines for comunication with Golog
- control procedures for the RCX side

It monitors for incoming messages from Golog requesting

the execution of an action or querying the occurrence of

exogenous events.

delivery.nqh, delivery.nqc

Application dependent part of the NQC code.

delivery.nqh:

Defines constants

required by the

send/receive functions

in control.nqh for

communicating with

Golog

delivery.nqc:

contains code for all the behaviours and

code that monitors for the occurrence of

exogenous actions:

- void initialize()
- void startBehaviour(int num)
- void stopAllBehaviours()
- void panicAction()
- void turnAround() → added

5. Besides Golog

- Legolog Status
- Summary
- What’s new

Legolog status

- Implementation
  - Linux
    - SWI-Prolog
    - ECLiPSe Prolog (version 4.2 onwards)
  - Windows/MS-DOS
    - LPA DOS-Prolog (version 3.83)
- Availability
  - http://www.cs.toronto.edu/~cogrobo/Legolog/

Summary

- Facilitation of quick and easy experimentation with cognitive robotics ideas such as sensing, exogenous actions, concurrency, etc.
- Substitute Golog planner easily
- Port to another Prolog/operating system realtively easy (provided accessible serial port)
- Problems:
  - Packet corruption in LEGO protocol
  - Checking for exogenous actions dependent on planner

What's new

An smodels-based version of the controller is available from M^2AG for experiments and/or thesis work