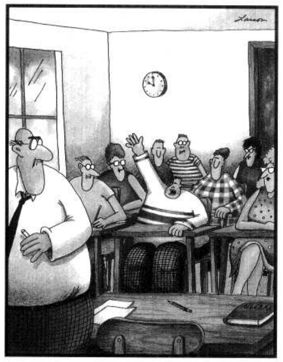
Legolog

Maurice Pagnucco
School of Computer Sc. & Eng.
University of New South Wales
NSW 2052, AUSTRALIA

morri@cse.unsw.edu.au
http://www.cse.unsw.edu.au/˜morri/

Legolog



"Mr. Osborne, may I be excused?
My brain is full."

Overview

- Introduction
- LEGO® MINDSTORMS™ RIS
- Legolog: The Basic Idea
- Prolog/RCX Communication
  - RCX User messages
  - Legolog Protocol
- NQC Code
- Delivery Task
- Using Alternatives to Golog
- Legolog Status
- Summary

Introduction

- Experimenting with cognitive robotics remains prohibitive due to the cost and maintenance of hardware, low-level issues, etc.
- LEGO® have introduced MINDSTORMS™ Robotics Invention System™ (RIS) construction kit equipped with programmable microprocessor that can accept input and control outputs
- Cost: approx $US 200
- **Aim:** provide a (Prolog-based) system for use in cognitive robotics research/teaching with effectors, sensors, exogenous events, concurrency, interrupts, . . .
- Use of Golog was our primary motivation however Golog can be easily substituted

Golog

- High-level programming language for intelligent agents
  - Based on Situation Calculus
  - Supports: sequence, conditionals, loops, non-deterministic choice; concurrency, priorities, interrupts, exogenous actions, sensing
  - Primitive statements—domain-dependent actions to be executed by 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

Golog—Programming Constructs

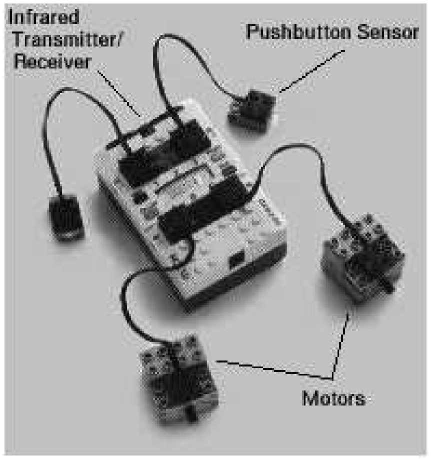
α	primitive action
$\phi?$	condition (wait)
$(\delta_1;\delta_2)$	sequence
if ϕ **then** δ_1 **else** δ_2 **endIf**	conditional
while ϕ **do** δ **endWhile**	loop
proc $\beta(\vec{x})\delta$ **endProc**	procedure definition
$\beta(\vec{t})$	procedure call
$(\delta_1 \mid \delta_2)$	nondeterministic choice of actions
$(\pi\, \vec{x})[\delta]$	nondeterministic choice of arguments
δ^*	nondeterministic iteration
$(\delta_1 \| \delta_2)$	concurrent execution
$(\delta_1 \rangle\rangle \delta_2)$	prioritised concurrency
$\delta^{\|}$	concurrent iteration
$\langle x : \phi \rightarrow \delta \rangle$	interrupt
search(δ)	search

LEGO® MINDSTORMS™ RIS

RCX (Robotic Command Explorer)

- Hitachi H8/3297 microprocessor
- 3 inputs
  - pushbutton, light, temperature, rotation
- 3 outputs
  - motors, light
- Infrared communications port allowing communication with infrared tower attached to serial port of personal computer
- Programming: LEGO®, NQC, LegOs, plus many more
- **Idea:** write program on standalone computer and download to RCX

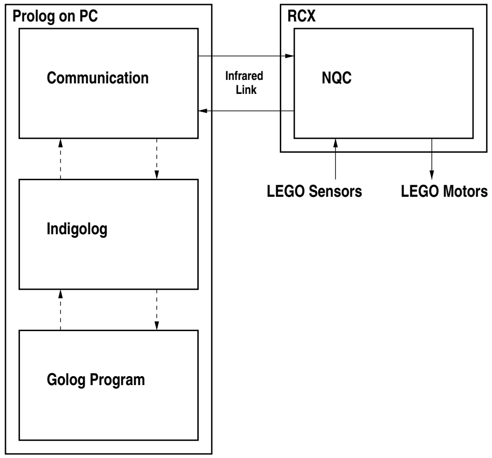
LEGO MINDSTORMS RIS



Legolog: The Basic Idea

- Written in Prolog and NQC
- Communicates actions via infrared tower
- Prolog initiates all communication
  - Golog determines next action to execute and sends message to RCX; RCX must acknowledge within 3.5 seconds with sensing value
  - Golog can also “query” RCX to determine whether exogenous action has occurred (currently, only one exogenous action stored)
- Using Indigolog interpreter: concurrency, interrupts, exogenous actions, search operator

Legolog



RCX User Messages

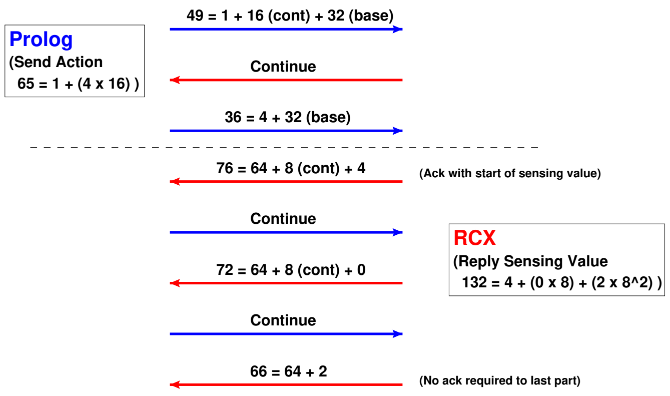
- RCX has simple error-checking protocol for communicating via infrared transmitter/receiver
- Messages are used to program RCX firmware, check battery level, etc.
- One particular message type—user message (our terminology)—allows numbers in the range 1 – 255 to be sent/received
- Legolog uses these for all communication
- User message packet format

85	255	0	247	8	MesgNo	255 - MesgNo	Checksum	255 - Checksum

Legolog Protocol

- **Desideratum:** send/receive arbitrarily large (positive) numbers
  - Allow multiple RCXs
  - Arbitrary sensing values
- How?
  - Send numbers $1 \leq n \leq 7$ bits at a time (least significant bits first)
  - Make use of a “continuation bit” to signal that more information is to follow
  - Also, a handful of special messages (exogenous request, continue, abort, request extra time, no exogenous action)
- Prolog initiates all communication (due to infrared tower “time-out”)
  - Not a problem since RCX would need to wait for Golog anyway

Legolog Protocol



Actions: base = 32; 4 bits per message
Sensing values: base = 64; 3 bits per message

NQC Code

- LEGO® provides firmware—virtual machine that can be downloaded to and run on RCX
- Not Quite C (NQC) is an independent C-like programming language for programming firmware (Baum, 2000)
- For Legolog need to provide
  - initialize: initialise RCX, start exogenous action monitors, etc.
  - startBehaviour: determine which behaviour to perform on input
  - panicAction: what to do when Prolog not responding to RCX
  - Plus code for behaviours, exogenous event monitoring, functions, etc.
- Actions possibly taking long time to execute can be dealt with in two ways
  - Transform into clipping actions: a start action and an exogenous action signalling completion
  - Request additional 3.5 seconds

NQC main loop

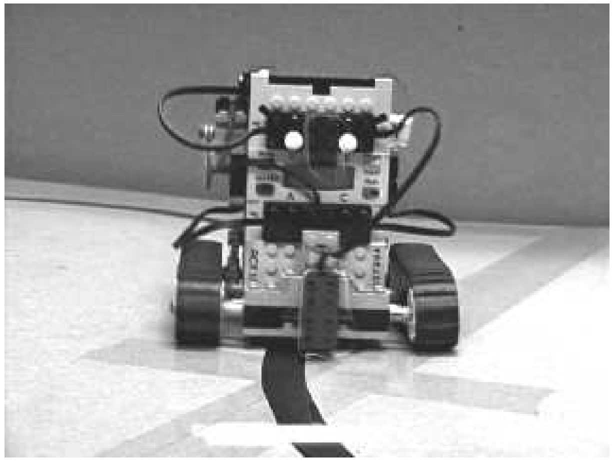
```
initialize();
while (true) {
    if (status == ABORT) {
        stopAllBehaviours();
        status = OK;}
    if (status == PANIC) {
        panicAction();                          //Move around, wiggle, beep,whatever
        SendMessage(PANIC_MESG);
        ReceiveMessage(result);}                //Hope for an abort command
    if (status == OK) {
        ReceiveMessage(result);
        if (validActionMesg(result)) {
            startBehaviour(result);
            SendMessage(sensingValue);}         //Return sensor value
         else if (exogRequestMesg(result)) {
            SendMessage(exogAction);
            exogAction = NO_EXOG_ACTION;}}}
```


Delivery Task

- Golog program

  ⟨motion = Lost → (recover); *start_to_next_station*⟩⟩
  ⟨motion = Moving → *wait*⟩⟩
  ⟨StopRequested(location) → *signal_arrival; wait*⟩⟩
  ⟨n: NextLocationToServe(n) →
  if location $<$ n **then** *Head_to_next_station(1)*
  else *Head_to_next_station*(-1)⟩⟩⟩
  ⟨location $>$ 1 → *Head_to_next_station*(-1)⟩⟩⟩
  ⟨**true** → *wait*⟩

Delivery Robot



Using Alternatives to Golog

- Prolog
  - Retain low-level Prolog implementation dependent code and RCX communication primitives
  - Supply new planner
    - initializeRcx/0: initialise serial port
    - actionNum/2: action/number mapping
    - sendRcxActionNumber/2: execute action and obtain sensing result
    - receiveRcxActionNumber/3: exogenous actions
    - finalizeRcx/0: tidy up
- RCX
  - If new planner cannot deal with exogenous actions, alter behaviours to request additional time

Legolog Status

- Implementation
  - Linux
    - SWI-Prolog
    - ECLiPSe Prolog (version 4.2 onwards)
  - Windows/MS-DOS
    - LPA DOS-Prolog (version 3.83) on HP200LX
- Availability

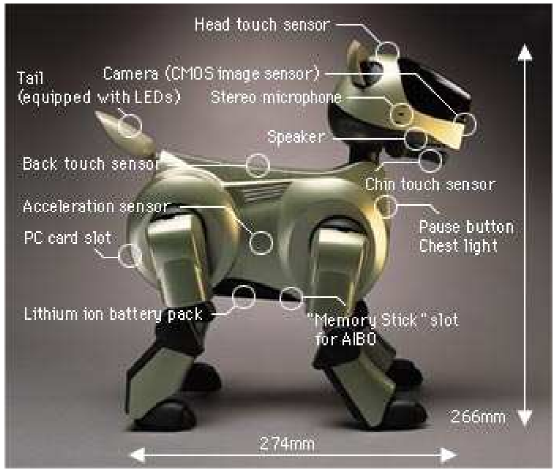
  `http://www.cs.toronto.edu/~cogrobo/Legolog/`

Summary

- Facilitation of quick and easy experimentation with cognitive robotics ideas such as sensing, exogenous actions, concurrency, etc.
- Allows for multiple robots—additional Golog constructs to make task easier
- Possible constructions—vast!
- Substitute Golog planner easily
- Port to another Prolog/operating system relatively easy (provided accessible serial port)
- Problems
  - Packet corruption in LEGO® protocol
  - Checking of exogenous actions dependent on planner
- Available from:

  `http://www.cs.toronto.edu/~cogrobo/Legolog/`

Sony ERS-2100



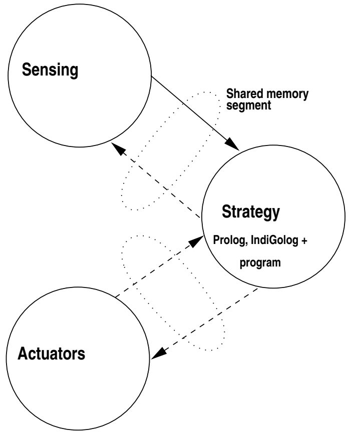
Sony ERS-2100

- CPU — 64-bit MIPS RISC
- Sensors
  - CMOS camera in head
  - head, chin, back, leg pressure sensors
  - temperature, infrared, acceleration, vibration sensors
  - microphone
- Actuators
  - legs, head, tail, ears
  - 20 degrees of freedom
  - speaker, LEDs

Desiderata

- High-level language for describing player strategy
- Still allow access to low-level data elements and actions
- Clearly separate strategy from lower level code
- Ability to rewrite strategy quickly and easily
- Deliberation for better action selection and "longer-term" planning
- However, require real-time interaction
- Inter-agent communication
- Interface should allow for other languages to be used to describe high-level strategy (i.e., not dependent on one approach)
- Currently looking at implementing ball collection challenge in Golog; have implementation in Prolog

UNSW Aperios Code Structure



Problems/Issues with IndiGolog

- Doesn't explicitly cater for real-time interaction
- Actions with duration
- Uninterruptable search
- Exogenous actions invalidate search
- Noisy sensors; unpredictable actuators
- Concurrent actions
- Low-level variables being updated every 1/25th second, how do we incorporate these changes into Golog
  - not all changes may be significant

Prolog in Strategy/Deliberative Object

- Due to memory available require (very) small Prolog implementing core functionality
- Must run on robot
  - wireless link is unpredictable
  - in any case want self-contained robot
- Use iProlog
  `http://www.cse.unsw.edu.au/~claude/research/prolog.html`
- ISO Prolog
- IndiGolog interpreter runs in Prolog

Information at our Disposal

- x, y, θ + variance for:
  - robot (self)
  - ball
  - teammate(s)
  - opponent(s)
  - own goal
  - opponent goal

Information at our Disposal

- x, y, θ_{rel}, $dist$ + variance for:
  - vision ball
  - vision own goal
  - vision opponent goal
- Other variables:
  - previous attack mode
  - robot state

Actions

- *dog_stand*
- *dog_head_find_ball*
- *dog_track_ball*
- *dog_semi_circle_find_ball*
- *dog_full_circle_find_ball*
- *dog_go_to_position_heading(x, y, theta)*
- *dog_go_hold_ball*
- *dog_kick(type, power, direction)*
- *dog_find_opponent_goal*

Communication between Layers

- Vision layer and Strategy layer
  - Vision variables copied to shared memory
  - Message sent to deliberative layer informing of update
- Strategy layer and Actuator layer
  - Strategy layer copies action to perform or position of joints to shared memory
  - Actuator layer continuously checks for changes in variables and takes necessary action

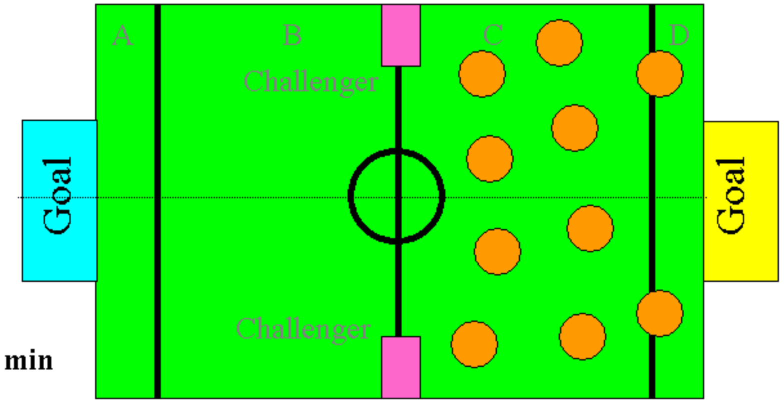
Golog Fluents

- As above plus
  - state = {goal_located, have_ball, found_ball, lost_ball}
  - Own_area(x, y)
  - Close_enough_own_goal(x, y)
  - Close_enough_opponent_goal(x, y)

Primitive Actions

- As above, e.g.,
  - *dog_full_circle_find_ball*
  - *dog_go_hold_ball*
  - *dog_kick(type, power, direction)*
- Also possible to define actions by giving position of actuators (this is also true of Prolog)
- Note that it is possible to perform actions concurrently on ERS-2100.

Ball Collection Challenge



3 min

Sample High-Level Program

proc *Control*
  dog_full_circle_find_ball;
  while $(\exists n)$ Ball(n) **do**
    dog_go_hold_ball;
    Find_goal;
    Shoot_ball;
    dog_full_circle_find_ball
  endwhile
endProc

proc *Find_goal*
  dog_find_own_goal | *dog_find_opponent_goal*
endProc

proc *Shoot_ball*
  if (Close_enough_own_goal(my_x_pos, my_y_pos))
    then *dog_kick*(CHEST_PUSH, MAX_POWER, STRAIGHT)
    else *dog_kick*(GOALIE_KICK, MAX_POWER, STRAIGHT)
endProc

Sample Golog Program

proc *Control*
  while (**true**) **do**
    if(state = lost_ball)
      then *dog_full_circle_find_ball*;
      else if (state = found_ball)
        then *dog_go_hold_ball*;
        else if(state = have_ball)
          then *Find_goal*(my_x_pos, my_y_pos)
          else *Select_kick*(my_x_pos, my_y_pos)
  endWhile
endProc

Sample High-Level Program

proc *Control*
  $\langle$state=goal_located $\rightarrow$ *Select_kick*$(my\_x\_pos, my\_y\_pos)\rangle\ \rangle\rangle$
  $\langle$state=have_ball $\rightarrow$ *Find_goal*$(my\_x\_pos, my\_y\_pos)\rangle\ \rangle\rangle$
  $\langle$state=found_ball $\rightarrow$ *dog_go_hold_ball*$\rangle\ \rangle\rangle$
  $\langle$**true** $\rightarrow$ *dog_full_circle_find_ball*$\rangle$
endProc

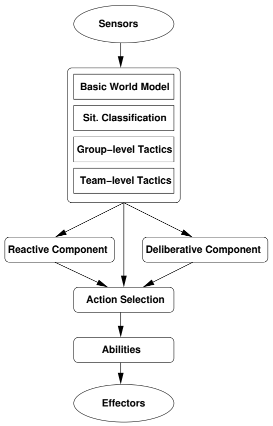
proc *Find_goal*(x, y)
  else if (Own_area(my_x_pos, my_y_pos))
    then *dog_find_own_goal*
    else *dog_find_opponent_goal*
endProc

proc *Select_kick*(x, y)
  if (Close_enough_own_goal(x, y) **or** Close_enough_opponent_goal(x, y))
    then *dog_kick*(CHEST_PUSH, MAX_POWER, STRAIGHT)
    else *dog_kick*(GOALIE_KICK, MAX_POWER, STRAIGHT)
endProc

Other Work

- Aachen University of Technology
  - deliberative/reactive architecture for robot soccer
  - focusses on middle-size and simulator league
- University of Melbourne – RoboMutts
  - Smalltalk to implement high-level strategy
- University of Freiburg — extended behaviour networks
- University of Koblenz-Landau — RoboLog
  - Prolog interface to RoboCup simulator
- Sabeena Chelat, Macquarie University — implementation of simple passing strategies in Golog for RoboCup simulator

Other Work — Aachen



Conclusions

- Work is in its infancy. To date have spent much time porting a small Prolog interpreter, cleaning up code and implementing interface between layers
- Can execute primitive actions using Golog
- Re-written ball collection challenge in Golog
- Experimenting with different modes of interaction with lower level
- No real deliberation to speak of as yet
- Variants of Golog (e.g., DTGolog), execution monitoring
- Much more work to be done!