What is Cognitive Robotics?

According to [Reiter01b]:

Cognitive robotics is the theory and implementation of robots/agents that reason, act and perceive in a changing, incompletely known, unpredictable environment.
Reasoning about:

- goals,
- actions,
- when to perceive and what to look for,
- the cognitive states of other agents.

Cognitive robotics is concerned with integrating reasoning, perception and action within a uniform theoretical and implementation framework.

Cognitive Robotics: An Overview

Yves Lespérance

Department of Computer Science
York University
Toronto, Canada

http://www.dis.uniroma1.it/~degiacomo/CogRobCourse01
http://www.cs.yorku.ca/~lesperan
http://www.cs.toronto.edu/~cogrobo

History of Robot/Agent Architectures (cont.)

- Hybrid architectures:
  - use both deliberative and reactive layers;
  - between planner/plan library and low-level controller, have sophisticated executor, e.g. RAP [Firby87], PRS [RaoGeo92].
- High-level programming:
  - between using a planner and writing a normal program;
  - task-level controller is program in very high-level language that uses a model/theory of application domain;
  - can write detailed plans;
  - can also write nondeterministic programs that require the interpreter to search, a form of planning;
  - e.g. Golog [LRLLS97].

History of Robot/Agent Architectures [WooJen95]

- Mainstream robotics:
  - focus on sensing, path planning, and low-level control;
  - task-level controller = C program or user.
- Deliberative architectures:
  - task-level controller = planner + simple executor;
  - use logic-based representations;
  - e.g. Shakey.
- Reactive architectures:
  - decomposition according to behavior;
  - each behavior responsible for sensor data interpretation;
  - no representation;
  - e.g. Brooks's subsumption architecture [Brooks91].

Situation Calculus [McCarthy63]

A language of predicate logic for representing dynamically changing worlds.

The constant S_0 represents the initial situation.

The term $do(\alpha, s)$ represents the situation that results from primitive action α being performed in situation s.

Predicates and functions whose value varies from situation to situation are called fluents.

e.g.

$$\forall o \, \neg Holding(o, S_0)$$

$$Holding(O_1, do(pickUp(O_1), S_0))$$

The Toronto Cognitive Robotics Framework

Uses:

- domain theory expressed in situation calculus;
- high-level programming languages that extend theory and allow complex actions/processes to be specified:
  - Golog = Algol + nondeterminism,
  - ConGolog = Golog + concurrency,
  - IndiGolog = ConGolog + incremental execution,
  - etc.
- implemented in Prolog.

Specifying a Domain in the Situation Calculus (cont.)

- Axioms describing the initial situation, S_0, e.g.

$$robotPlace(S_0) = CentralOffice.$$

- Unique names axioms for the primitive actions.
- Some foundational, domain independent axioms [LinRei94].

This is called a *basic action theory* [Reiter01b].

Specifying a Domain in the Situation Calculus

Use a theory that includes:

- Action precondition axioms, one for each primitive action α, which characterizes $Poss(\alpha, s)$, e.g.

$$\begin{array}{l} Poss(pickUpShipment(n), s) \equiv \\ \quad orderState(n, s) = ToPickUp \wedge \\ \quad robotPlace(s) = mailbox(sender(n, s)). \end{array}$$

- Successor state axioms, one for each fluent F, which characterize the conditions under which $F(\vec{x}, do(a, s))$ holds in terms of what holds in situation s; these axioms may be compiled from effects axioms, but provide a solution to the frame problem [Reiter91], e.g.

$$\begin{array}{l} robotDestination(do(a, s)) = p \equiv \\ \quad a = startGoTo(p) \vee \\ \quad p = robotDestination(s) \wedge \\ \qquad \forall p\, a \neq startGoTo(p) \end{array}$$

Golog [LRLLS97]

AlGOl in LOGic

Constructs:

α,	primitive action
$\phi?$,	test a condition
$(\delta_1 ; \delta_2)$,	sequence
if ϕ **then** δ_1 **else** δ_2 **endIf**,	conditional
while ϕ **do** δ **endWhile**,	loop
proc $\beta(\vec{x})$ δ **endProc**,	procedure definition
$\beta(\vec{t})$,	procedure call
$(\delta_1 \mid \delta_2)$,	nondeterministic choice of action
$\pi\, \vec{x}\, [\delta]$,	nondeterministic choice of arguments
δ^*,	nondeterministic iteration

Example Application: Robot Mail Delivery

Task-level controller must produce and execute delivery plans.

Must react to events such as:

- new shipment orders,
- navigation failures.

Planning useful for:

- finding optimal route,
- dealing with failures,
- dealing with unexpected requests.

A Very Simple Golog Robot Control Program

Does mail delivery. Serves orders randomly.

proc $control$
  while $\exists o\, Order_to_serve(o)$ **do**
    $\pi o\ [Order_to_serve(o)?;$ % choose an order
      $serve_order(o)]$ % and serve it
  endWhile
endProc

proc $serveOrder(o)$
  $go_to(sender(o)); pickUp(o);$
  $go_to(recipient(o)); dropOff(o)$
endProc

Golog (cont.)

Programmer provides a basic action theory to specify primitive actions and what is true initially.

Semantics: $Do(\delta, s, s')$, meaning that program δ starting in situation s may terminate in situation s'; because of nondeterminism, can be many s'; Do defined in sit. calc.

Interpreter searches for a final situation s' for program; sequence of actions in s' is a possible execution.

Can be executed for real if programmer provides implementation for primitive actions.

Using Nondeterminism to Do Planning: A Mail Delivery Example

This control program searches to find a schedule/route that serves all clients and minimizes distance traveled:

proc $control$
$minimize_distance(0)$
endProc

proc $minimize_distance(distance)$
$serve_all_clients_within(distance)$
| % or
$minimize_distance(distance + Increment)$
endProc

$mimimize_distance$ does iterative deepening search.

Using Nondeterminism: A Simple Example

A program to clear blocks from table:

$$(\pi\, b\, [OnTable(b)?; putAway(b)])^*; \neg\exists b\, OnTable(b)?$$

When condition of a test action or action precondition is false, backtrack and try different nondeterministic choices.

Interpreter will find way to unstack all blocks ($putAway(b)$ is only possible if b is clear).

Interpreter searches all the way to a final situation of the program, and only then starts executing corresponding sequence of actions.

ConGolog [DLL00]

Extends Golog with constructs to support concurent programming:

$(\delta_1 \parallel \delta_2)$,	concurrent execution
$(\delta_1 \rangle\!\rangle \delta_2)$,	concurrency with priorities
$\delta^{\parallel}$,	concurrent iteration
$\langle \vec{x} : \phi \rightarrow \delta \rangle$,	interrupt

Concurrency makes it easy to write more reactive controllers.

A Control Program that Plans (cont.)

proc $serve_all_clients_within(distance)$
$\neg\exists c\; Client_to_serve(c)$?
% if no clients to serve, we're done
| % or
$\pi c, d$ [($Client_to_serve(c) \wedge$ % choose a client
$d = distance_to(c) \wedge d \leq distance$?);
$go_to(c)$; % and serve him
$serve_client(c)$;
$serve_all_clients_within(distance - d)$]
endProc

ConGolog (cont.)

Test action ϕ? now interpreted as "wait for condition ϕ".

When condition of wait action or action precondition is false, process blocks and other processes can execute.

If no process can execute and not done, backtrack.

Interpreter still searches all the way to final situation/end of program, before executing any actions. Not practical for long programs. May lack knowledge to find a plan.

Simple ConGolog Example: Reactive Robot Mail Delivery

proc $serveOrder(o)$
  $goTo(sender(o)); pickUp(o);$
  $goTo(recipient(o)); dropOff(o)$
endProc;

$\langle\, o : OrderToServe(o) \rightarrow serveOrder(o)\,\rangle$
$\gg$
$\langle\, robotPlace \neq Home \rightarrow goTo(Home)\,\rangle$

Always execute highest priority process that is not blocked.

When interrupt gets control and its condition is true, it triggers and body is executed.

Once execution is completed, interrupt can trigger again.

IndiGolog (cont.)

Program can include *sensing actions* that acquire new information:

- result of sensing added to basic action theory,
- programmer must provide method to get sensing result.

Changes in environment can be detected as *exogenous actions:*

- interpreter monitors for them and adds to history,
- programmer must provide method to check occurrence.

IndiGolog [DegLev00]

Variant of ConGolog for incompletely known, dynamic environments.

Supports controllers that interleave sensing, planning, and plan execution.

By default, no search/lookahead.

But, programmer can request interpreter to *search* over block of code using new construct:

$\Sigma\ \delta$ search block

Search is done on-line!

Reactivity and Search

May need to react quickly to exogenous actions.

E.g. acknowledging new orders.

[LesNg00]'s extensions allow combining reactive threads & planning threads that do search:

proc $control$
$< \; o : \; New_order(o) \rightarrow ack_order(o) \; >$
$\rangle\!\rangle$
$\Sigma(minimize_distance(0))$
end

Acknowledgement will be done before replanning!

Limitation: planning will not be interrupted.

Need to keep track of actions that come from inside search block.

IndiGolog Replanning

When an unexpected exogenous action happens while executing a search block, may need to *replan*.

E.g. when running mail delivery program that minimizes distance travelled and new shipment order is made.

Then, IndiGolog checks if the sequence of actions found earlier is still an execution of the program in the search block; otherwise, it redoes the search.

Original IndiGolog of [DegLev00] can't find a plan for this e.g. because it restarts search from current program.

IndiGolog of [LesNg00] restarts search from original program, so it does find a new plan.

Only committed to the actions it has performed.

Relying on Exogenous Actions
E.g. Coping with Navigation Failures (cont.)

[LesNg00]'s extended IndiGolog supports this.

During planning, use *simulated environment modeled as program:*

proc $control$
  $\Sigma(minimize_distance(0) \parallel env_simulator)$
endProc

proc $env_simulator$
  $< robot_state = Moving \rightarrow \mathbf{sim}(reach_dest) >$
endProc

Simulator and planning process assumes navigation will always succeed.

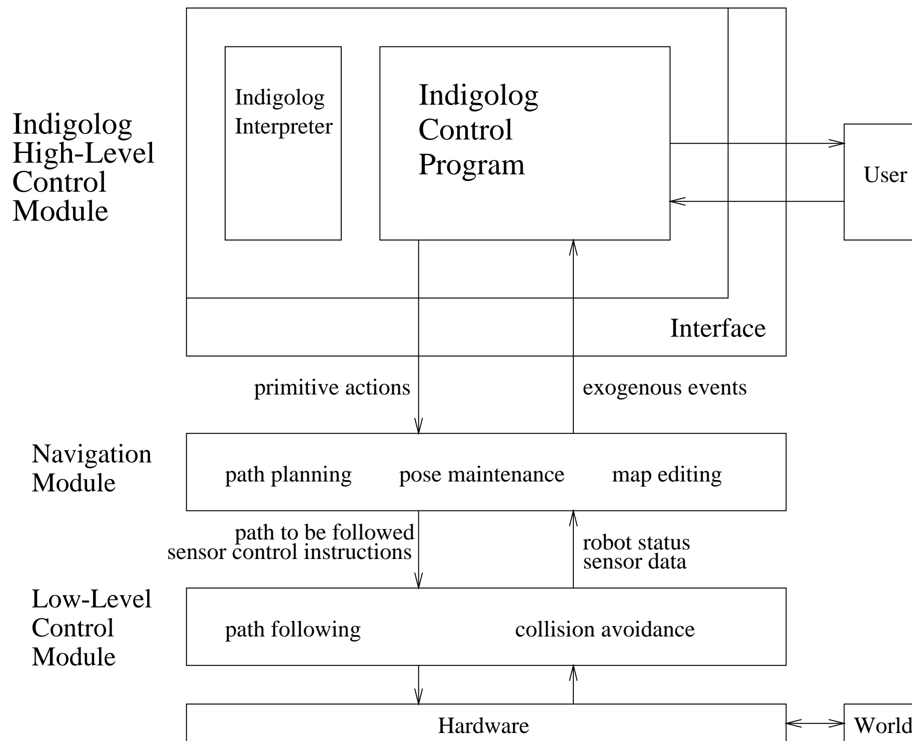
If navigation fails, replanning occurs.

Relying on Exogenous Actions
E.g. Coping with Navigation Failures

Suppose controller is notified of success by $reachDest$ exogenous action and of failure by $getStuck$ exogenous action:

proc $serve_all_clients_within(distance)$
  $\neg\exists c\ Client_to_serve(c)?$ |
  $\pi c, d$ [($Client_to_serve(c) \wedge$
      $d = distance_to(c) \wedge d \leq distance)?$;
    $start_goto(c)$; % start to navigate to client
    $robot_state \neq Moving?$; % wait until robot stops
    if $robot_state = Reached$ **then**
      $serve_client(c)$;
    else
      $handle_service_failure(c)$;
    endIf
    $serve_all_clients_within(distance - d)$]
endProc

York's Hierarchical Control Architecture



Relying on Exogenous Actions E.g. Coping with Navigation Failures (cont.)

During planning, treat *simulated actions* as real actions.

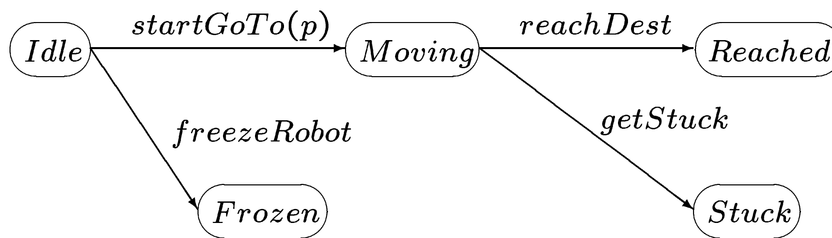
But simulated actions are never executed. Interpreter waits for a real exogenous action to occur.

If exogenous action is as expected then continue, else replan.

But so far, simple deterministic environment simulators only!

Interfacing H-L Controller, E.g. model of [LTJ99]

From navigation point of view, model robot as being in a state $robotState(s)$ which can change as a result of actions by the high-level controller and exogenous events:



Also action $resetRobot$ that returns robot to $Idle$ state; can be performed in any state. Also fluents $robotDestination(s)$ and $robotPlace(s)$.

So treat "going to a location" as an "activity" in the sense [Gat92]; the primitive actions are not the activity; only initiate and terminate it.

Interfacing High-Level Controller with Rest of Architecture

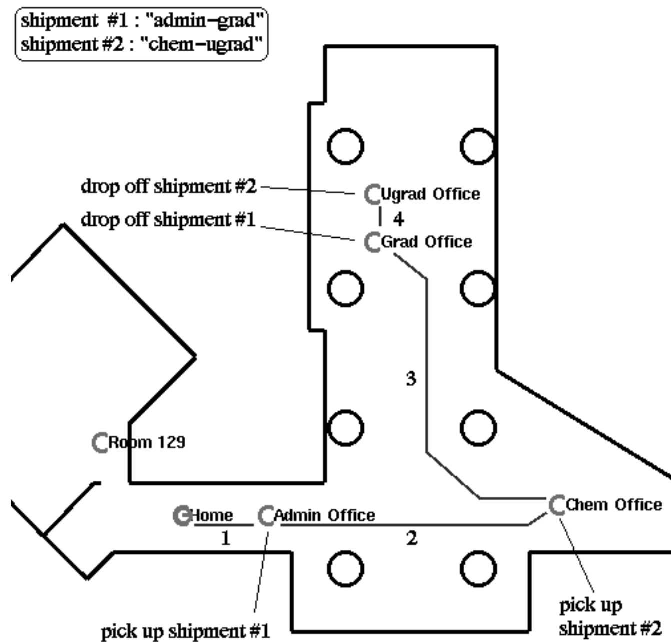
One approach: view rest of architecture as another agent; primitive actions are commands to it and signals it sends back are exogenous actions.

In high-level controller, use model of rest architecture.

Simple version of this in [LTJ99].

Other work on this problem: [FinPir01], and [GroLak00] on cc-Golog for continuous control.

E.g. [LesNg00] Test Scenario 1: Planning Optimal Route



Implementation and Experimentation

Controllers written in situation calculus-based high-level programming languages tested on real robots at York, U. of Toronto, and U. of Bonn; Bonn group created very successful “museum guide” application [Bur+98].

At York: high-Level controllers that do mail delivery and handle new orders and navigation failures; run on RWI B12 and Nomad Super Scout.

Use low-level control and navigation modules based on software developed for ARK project [Nic+98]. Modules run in separate processes that communicate via TCP/IP sockets.

In [LesNg00], system using extended IndiGolog tested on scenarios that require planning to optimize delivery route, reacting/replanning when new order arrives or navigation fails.

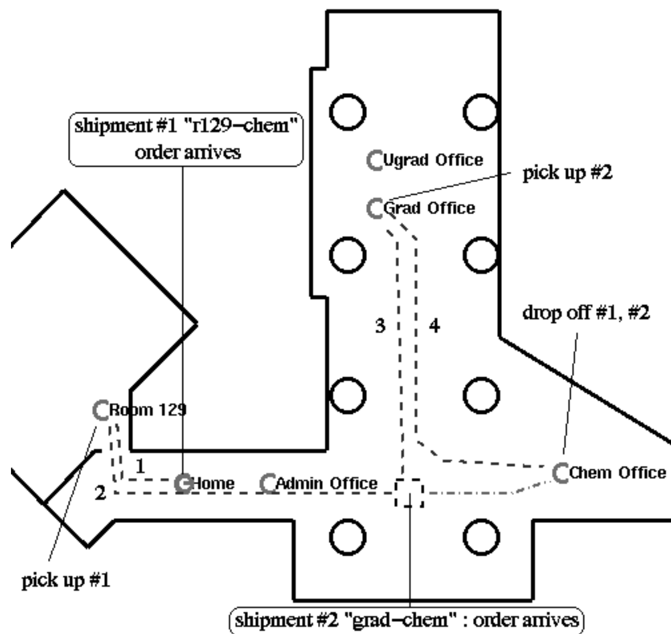
Recap

High-level programming in situation calculus is promising approach to cognitive robotics

IndiGolog high-level programming language supports:

- complex behaviors: loops, concurrency, etc.,
- reactive behaviors: interrupts,
- on-line planning,
- execution in dynamic & incompletely known environments.

E.g. [LesNg00] Test Scenario 2: Replanning to Serve Urgent Order



Current Issues in Cognitive Robotics Research

Representing and reasoning about action's effects and preconditions, ramification and qualification problems [Reiter01b].

High-level programming with temporal constraints [Reiter98].

Modeling & interfacing with non-symbolic processes, continuous control [GroLak00], [Sandewall97].

Representing and reasoning with incomplete knowledge [BacPet98], [DemDel00].

Accounts of sensing and knowledge change [Iocchi99], [ShaWit00], [MciSch00], on-line sensors and just-in-time programs [DLS01], noisy sensors [BHL95], models of vision, anchoring [CorSaf01].

Recap (cont.)

Logical foundations support:

- formal specification and verification,
- reasoning with *incomplete* information.

Tested in simple but real robotics applications.

Also high-level programming in event calculus [ShaWit00] and fluent calculus [Thielscher00], and related work on Model-Based Programming [WCG01] and structured-reactive controllers [Beetz01].

Current Issues in Cognitive Robotics Research (cont.)

Richer accounts of mental states, belief, goals, etc. [SPLL00], [ShaLes01], ability [Levesque96], [LLLS01].

Representing and reasoning about other agents/robots [ShaLes01].

Infrastructure for multirobot/agent systems.

Coordination protocols.

Coordination planning.

Adversarial planning.

Development methodologies.

Verification.

Emotions, Personality, etc.

Current Issues in Cognitive Robotics Research (cont.)

Planning with incomplete information, conformant, conditional [BacPet98], [FPR00], [Sardina01], knowledge-based programs [Reiter01a].

Contingent planning.

Agent architecture, integrating sensing, planning, and plan execution, execution monitoring [DRS98].

Probabilistic modeling [FinPir01], [GroLak01], [BHL95].

Decision-theoretic modeling [BRST00], [WCG01].

Practical use of planning as nondeterministic programming.