COGNITIVE ROBOTICS

Raymond Reiter
Department of Computer Science
University of Toronto
and
The Canadian Institute for Advanced Research

Being a progress report for the U of T Cognitive Robotics Group:

Yves Lesperance	Hector Levesque
Fangzhen Lin	Richard Mann
Daniel Marcu	Ray Reiter
Richard Scherl	

ftp://ftp.cs.toronto.edu:/pub/cogrob/README

Cognitive Robotics

- Theory and implementation of agents that reason, act and perceive in changing, incompletely known, unpredictable environments. Reasoning about:
  - Goals.
  - Actions.
  - Time, continuous events.
  - When to perceive and what to look for.
  - The cognitive states of other agents.
- In short, cognitive robotics is concerned with integrating reasoning, perception and action within a uniform theoretical and implementation framework.
- Contrast with virtually all current robotics research.
  - Sensory processing.
  - Path planning.
  - Manipulator design and control.
  - Reactive agents, cockroach AI.
- Requires a sophisticated theory of actions, including perceptual actions, complex actions, concurrency, exogenous events, etc.

This Talk

- Suitably enriched situation calculus as a foundational language for theories of actions.
  - Start with a theory of simple (primitive) actions.
  - Solve the *frame problem* for these.
  - Bootstrap to a situation calculus based theory for complex actions.
  - This leads to GOLOG, a high level logic programming language for robot control.
  - Augment the situation calculus with the ability to talk about an agent's state of *knowledge* about its environment.
  - Enrich this formalism with perceptual actions, whose effects are to increase an agent's knowledge about its environment.
    $\Longrightarrow$
    Integration of reasoning, action and perception within a uniform theoretical and implementation framework.
  - This is the foundation for our theory and programming for cognitive robotics.
- Fruitful connections with other areas of computer science (database theory, software specification, programming languages).

The Situation Calculus (McCarthy, 1963)

- A first order language for representing dynamically changing worlds; all changes are the result of named $actions$.
- The world is conceived as being in a state s; this state can change only by virtue of some agent performing an action.
- $do(\alpha, s)$ denotes the successor state to s resulting from performing the action α.
- Actions may be parameterized:

  $put(x, y) =$ put object x on object y.

  $do(put(A, B), s) =$ that state resulting from placing A on B when the world is in state s.
- $Fluent =$ relation or function whose values may vary from state to state.
  - Denoted by predicate/function symbol taking a state term as one of its arguments.

    $closeTo(x, y, s)$: x is close to y when the world state is s.

    $pos(x, s)$: x's position when the world state is s.

The Situation Calculus (Continued)

- Actions have *preconditions*:

$$Poss(pickup(x), s) \equiv \\ [(\forall z)\neg holding(z, s)] \wedge nexto(x, s) \wedge \neg heavy(x).$$

$$Poss(repair(x), s) \equiv hasglue(s) \wedge broken(x, s).$$

- World dynamics are specified by *effect axioms*.
  - These specify the effects of a given action on the fluents – the causal laws of the world.

$$Poss(drop(x), s) \wedge fragile(x) \supset broken(x, do(drop(x), s)).$$

$$Poss(repair(x), s) \supset \neg broken(x, do(repair(x), s)).$$

$$Poss(paint(x, c), s) \supset colour(x, do(paint(x, c), s)) = c.$$

The Frame Problem (McCarthy and Hayes)

- Axioms other than effect axioms are required:
  - *Frame axioms.*
  - Specify the action *invariants* of the domain: those fluents unaffected by the action.
  - Dropping things does not affect an object's colour:

$$\begin{aligned} &Poss(drop(x), s) \wedge colour(y, s) = c \\ &\qquad\qquad \supset colour(y, do(drop(x), s)) = c. \end{aligned}$$

  - Not breaking things:

$$\begin{aligned} &Poss(drop(x), s) \wedge \neg broken(y, s) \wedge [y \neq x \vee \neg fragile(y)] \\ &\qquad\qquad \supset \neg broken(y, do(drop(x), s)). \end{aligned}$$

- *Frame Problem:*
  - Vast number of frame axioms; only relatively few actions will affect the truth value of a given fluent.
  - All other actions leave the fluent invariant.
  - An object's colour is not changed by picking things up, opening a door, turning on a light, electing a new prime minister of Canada, etc. etc.
- $\sim 2 \times \mathcal{A} \times \mathcal{F}$ frame axioms.

What Counts as a Solution to the FP?

- Suppose an axiomatizer succeeds in writing down **all** the effect axioms for some application domain:
  For each predicate fluent F and each action A which can cause F's truth value to change, an axiom of the form

  $$Poss(A, s) \wedge R(s) \supset \pm F(do(A, s)).$$

  For each functional fluent f and each action A which can cause f's value to change, an axiom of the form

  $$Poss(A, s) \wedge R(s) \supset f(do(A, s)) = y.$$

- Want a systematic procedure for generating, from these effect axioms, all the frame axioms.
- Also want a *parsimonious* representation for the frame axioms (because in their simplest form, there are too many of them).

The FP: A Simple Solution (Sometimes)

R. Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, 1991.

- Based on earlier ideas of Pednault, Haas and Schubert.
- Start with the effect (causal) axioms:

  $$Poss(drop(x), s) \wedge fragile(x, s) \supset broken(x, do(drop(x), s)).$$

  $$Poss(repair(x), s) \supset \neg broken(x, do(repair(x), s)).$$

  Perform a simple, efficient, syntactic transformation to obtain a *successor state axiom* for *broken*:

  $$Poss(a, s) \supset [broken(x, do(a, s)) \equiv \\ fragile(x, s) \wedge a = drop(x) \vee broken(x, s) \wedge a \neq repair(x)].$$

- Analogous to the Clark completion of a logic program.
- Yields simple, extremely compact axioms.
- Warning:
  - Applies to *deterministic, primitive* actions only.
  - Does not address the *ramification/qualification* problems. But see:

    F. Lin & R. Reiter. State constraints revisited. *J. of Logic and Computation*, Special issue on actions and processes.

Primitive, Deterministic Actions Only

- As yet, we have no constructs for complex actions; all actions are *primitive.*
- Conditional actions:

$$\textbf{if } car\_in\_driveway \textbf{ then } drive \textbf{ else } walk.$$

- Iterative actions:

$$clear\_table \triangleq \textbf{while } [(\exists block) ontable(block)] \textbf{ do } remove\_a\_block.$$

- Nondeterministic actions:

$$remove\_a\_block \triangleq$$
$$(\pi\ block)[pickup(block); put\_on\_floor(block)].$$

- Recursive actions: If *down* means move an elevator down one floor, define *d(n)*, meaning move an elevator down n floors.

$$\begin{array}{l} \textbf{proc}\ \ d(n) \\ \quad \textbf{if}\ \ n = 0\ \ \textbf{then}\ \ no\_op \\ \quad \textbf{else}\ \ down; d(n-1) \\ \textbf{end} \end{array}$$

A Theory of Complex Actions

- A way of defining complex actions in terms of "easy" (wrt the frame problem) *primitive actions*.
- Complex actions, defined in terms of the primitives, inherit their solution to the frame problem.
- Operators for defining complex actions:
  - Sequence (Do action A, followed by action B).
  - Test actions (Is p true in the current state?)
  - While loops.
  - Conditionals (**if then else**).
  - Nondeterministic choice of actions (Do A or do B).
  - Nondeterministic choice of arguments to actions (Kick something: $(\pi\ x)kick(x)$).
  - Procedures: (Giving names to complex actions; recursion).
- Must have a compositional treatment of the frame problem for complex actions.
- Results in a novel programming language for discrete event simulation and high level robot control.

SC Definitions of Complex Action Constructors

$Do(a, s, s')$ – Doing action a in state s leads to a state s'.

- Primitive actions a:

$$Do(a, s, s') \triangleq Poss(a, s) \wedge s' = do(a, s).$$

- Sequence:

$$Do([a_1; a_2], s, s') \triangleq (\exists s'').Do(a_1, s, s'') \wedge Do(a_2, s'', s').$$

- Nondeterministic actions 1:

$$Do([a_1 \mid a_2], s, s') \triangleq Do(a_1, s, s') \vee Do(a_2, s, s').$$

- Nondeterministic actions 2:

$$Do((\pi\ x)a(x), s, s') \triangleq (\exists x)Do(a(x), s, s').$$

- Etc. (Conditional actions, test actions, while loops, concurrency, interrupts, procedures, including recursion)
- These definitions are macros; complex actions are completely eliminated by macro expansion.
- So a program (complex action) is a formula of the situation calculus.
- Important point: Purely logical, situation calculus semantics. Extremely compact specification.

GOLOG: A Novel Logic Programming Language

- GOLOG = alGOl in LOGic.
- Situation calculus semantics (first order logic + some second order).
- The GOLOG interpreter is a theorem-prover.
- Background axioms:
  - Definitions of complex action constructors (**while** loops, sequence, etc.)
  - Initial state of the world:

    $$ontable(A, S_0),\ blue(B, S_0) \vee red(B, S_0),$$

  - Action precondition axioms, one for each primitive action.
  - Successor state axioms, one for each fluent. (These solve the frame problem for primitive actions.)
- Evaluating a complex action α:

  $$Axioms \vdash (\exists s) Do(\alpha, S_0, s).$$

- Use answer extraction on s to obtain an execution trace, representing the sequence of actions the agent must perform.
- cf. Green (1969) wrt plan synthesis.
  But *do not confuse this approach with planning!*
- cf. Prolog.

Using GOLOG: An Example Robot Simulation

- Primitive actions:
  $pick\_up(pkg),\ drop\_off(pkg),\ go\_to(loc)$
- High-level control specified by complex actions:

  proc $control$
      while $(\exists p)Requires\_Shipment(p)$ **do**
          $(\pi r)[(\exists p).Requires\_Shipment(p) \wedge recipient(p) = r]?;$
          $serve(r)$
  end

  proc $serve(r)$
      $grab\_all\_letters\_for(r);$
      if $(\exists p).Robot\_is\_Carrying(p) \wedge recipient(p) = r$ **then**
          $go\_to(loc(in\_box(r)));$
          $drop\_off\_all\_letters\_for(r)$
  end

  $\ldots$

- Execution: Pose

$$Axioms \vdash (\exists s)Do(control, S_0, s).$$

  If successful, use answer extraction on s to get:
  $s = do(a_n, \ldots do(a_2, do(a_1)) \ldots)$
  Then emit actions $a_1, a_2, \ldots, a_n$ to executor.

Mailworld

Why Does This Work?

- As everyone knows, general purpose theorem-proving is computationally hopeless; so why has this strategy been effective?
  - Successor state axioms (which solve the frame problem for primitive actions) have a special syntactic form:

    $$Poss(a, s) \supset [F(\vec{x}, do(a, s)) \equiv \phi(\vec{x}, a, s)].$$

    So fluents are like *definitions*.
  - All theorem proving in GOLOG use *regression* wrt these successor state axioms to reduce queries about the current state, e.g.

    $$\exists p \;\; Robot\_is\_Carrying(p) \wedge recipient(p) = r$$

    to a theorem proving task *about the initial state*.
  - Theories about the initial state tend to be very simple, e.g. sets of ground literals (= open world *databases*).
- So we have:

  GOLOG = database + definitions ≈ logic programs.
- In fact, we use Prolog as an implementation language for GOLOG, essentially to do the regression wrt successor state axioms.

Knowledge and Perception

- Thus far, we have (implicitly) assumed that action preconditions can be formulated in terms of what is *true of the world*.
- Certain actions have preconditions involving the agent's *state of knowledge* about the world.

  $$Poss(telephone(x), s) \equiv$$
  $$(\exists t).is\_telephone(t) \wedge nexto(t, s) \wedge \mathbf{Kref}(telephone\#(x), s).$$

  $\mathbf{Kref}(\tau, s) =$ In state s, the agent knows the referent of τ. The agent knows who/what τ is in state s.

  $\mathbf{Kref}(author\_of(Waverly), s)$,

  $\mathbf{Kref}(favourite\_book(Mary), s)$.
- Thus far, we have (implicitly) assumed that actions only affect the state of the world, e.g. picking up a block, painting a wall.
- Certain actions have no effect on the world, but do affect the agent's *state of knowledge* about the world, e.g. perceiving that there is a red block on the table, being told (hearing) that Hector is in Rome.
- Call these *knowledge-producing actions*.
- Objectives: provide a formal account of actions with knowledge preconditions, and of knowledge producing actions and their effects on the perceiving agent's mental state.
- Solve the frame problem for knowledge producing actions.

Knowledge in the Situation Calculus

- We follow Moore's (1985) approach: axiomatize the $S5$ accessibility relation in the SC.
- Let $K(s', s)$ mean that, when the agent is in situation s, as far as he is concerned, the world could be in situation s'. K is an *accessibility relation* on situations.
- For $S5$, K must be reflexive, symmetric, transitive.

$$(\forall s) K(s, s).$$

$$K(s, s') \supset K(s', s).$$

$$K(s, s') \wedge K(s', s'') \supset K(s, s'').$$

- Notation: $\mathbf{Knows}(\phi, s)$ = in situation s, the agent *knows* ϕ.
- This means that ϕ is true in all situations s' accessible from s.

$$\mathbf{Knows}(\phi, s) \triangleq (\forall s').K(s', s) \supset \phi[s'].$$

- Knowing who/what:

$$\mathbf{Kref}(t, s) \triangleq (\exists x)\mathbf{Knows}(x = t, s).$$

- Solve the frame problem for knowledge producing actions = Obtain a successor state axiom for the accessibility relation K.
- R. Scherl and H. Levesque. the frame problem and knowledge-producing actions. *Proc. AAAI-93*.

Programs with Knowledge-Producing Actions

- We can now introduce knowledge-producing (sense) actions, whose effects are to augment the agent's knowledge about her environment.
- $sense\_requests$: results in the robot knowing whether a mail shipment is required, e.g. by reading her email.
- Then we can write programs that appeal to such sense actions.

proc *control*
  $sense\_requests$;
  while $(\exists p) Requires\_Shipment(p)$ **do**
    $(\pi r)[(\exists p).Requires\_Shipment(p) \wedge recipient(p) = r]?$;
    $serve(r)$;
    $sense\_requests$
end

Some Computer Science Connections

- Any branch of Computer Science concerned with *dynamics* – and what isn't? – can profit from theories of actions in AI, and vice versa.
  - Computer animation.
  - Update transactions in databases.
  - Software specification.
  - Programming languages.

A Computer Science Connection: Databases

- Identify elementary database transactions with primitive actions, arbitrary transactions with complex actions.
- The result is a situation calculus formalization of database evolution which is isomorphic to our theory of actions.
- Database integrity constraints $\simeq$ State constraints.
  Verifying integrity constraints $\simeq$ Ramification problem.
- From a database perspective, classical STRIPS operators are relational algebra operators of a very simple form.
  $\Longrightarrow$
  Can generalize STRIPS by permitting arbitrary RA expressions.
  - These generalized STRIPS operators can be compiled from the situation calculus axioms for the domain.

  Foundations for STRIPS = classical relational database theory.

R. Reiter. On Specifying Database Updates. *J. Logic Programming*, to appear.

R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence Journal*, **64**, 1993.

F. Lin and R. Reiter. How (and why) to progress a database I: formal foundations. *Proc. KR'94*.

F. Lin and R. Reiter. How to progress a database II: the STRIPS connection. *IJCAI-95*, to appear.

A CS Connection: Software Specification

- Basic idea:
  - Formally specify a program before implementing it.
  - Formal specification languages suitable for proving properties of programs (VDM, Z, etc).
  - Prove that the implementation meets its specification.
- Specification says what the program should achieve, not how.
  - By describing the program's effects with pre/postcondition assertions.
    "If pre is true in a state prior to executing the program, $post$ will be true after."
  - Exactly the same form as effect axioms; program = action.
- A frame problem; the specification must indicate which relations on program states are *unaffected* by a particular program!
- Existing specification languages provide constructs for the user to describe the frame invariants.
  - An additional burden on the user.
  - Constructs are often inadequate.
- Solution: Use the SC, and existing solutions to the FP.

A. Borgida, J. Mylopoulos and R. Reiter. The frame problem in procedure specifications. *IEEE Trans. on Software Engineering*, to appear.

A CS Connection: Programming Languages

- Our definitions of complex actions = the situation calculus version of the *dynamic logic* definitions of program constructors.
- Hoare's logic for reasoning about programs is derivable from the situation calculus definitions of complex actions.
- Classical regression in AI planning $\simeq$ repeated application of Hoare's inference rule for primitive program instructions.
- Lifschitz: Very little interaction between the AI planning and programming language communities. Why?
  - AI has focussed on the properties of primitive actions with complex effects on the world.
    $\Longrightarrow$
    Emphasis on issues like the frame/ramification problems.
  - Programming language theory has focussed on complex actions constructed from simple primitive actions.
    $\Longrightarrow$
    Emphasis on nondeterminism, concurrency, proving properties of programs.
- Time to exploit the commonalities.
  - Apply techniques of programming theory for proving properties (termination, correctness) of complex actions for robots.
  - Concurrency, nondeterminism.

Other Relevant Work

- *Indexicals*: Robot-centred representations.

  Y. Lespérance and H.J. Levesque. Indexical knowledge and robot action – a logical account, *Artificial Intelligence*.

- Planning algorithms in the situation calculus.

  F. Lin. On goal ordering in planning. U of T tech report.

- *Knowing How:* Ascribing to an agent the ability to carry out a complex procedure.

  Y. Lespérance, H.J. Levesque, F. Lin and R. Scherl. Ability and Knowing How in the Situation Calculus. U of T tech report.

- *Temporal* reasoning.

  J. Pinto and R. Reiter. Reasoning about time in the situation calculus. To appear in *Annals of Math. and AI*.

  J. Pinto. Temporal reasoning in the situation calculus. Ph.D. Thesis, Univ. of Toronto, Jan. 1994.

- Noisy sensors/effectors.

  F. Bacchus, J.Y. Halpern, and H. Levesque. Reasoning about Noisy Sensors in the Situation Calculus. *IJCAI-95*, to appear.

- Multiple agents, concurrency, exogenous events.

  Lespérance-Levesque-Reiter, in progress.

- GOLOG for agent-oriented programming and softbots.

  Y. Lespérance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. Foundations of a Logical Approach to Agent Programming. *Actes des Troisièmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents*, Chambery-St-Badolph, France, March, 1995.

  D. Marcu, Y. Lespérance, H. Levesque, F. Lin, R. Reiter, and R. Scherl. Distributed Software Agents and Communication in the Situation Calculus. *Proc. ICC-95*.

- *Implementation:* Controlling a simulated, and eventually physical, robot.

Summary: Foundations for Cognitive Robotics

- A solution to the frame problem for primitive actions.
- An enriched ontology for the situation calculus.
  - Complex actions.
  - Knowledge and perceptual actions.
  - Indexicals.
  - (Continuous) time.
  - Concurrency, exogenous events.
  - Noisy sensors/effectors.
- CONGOLOG (interleaving concurrent GOLOG), a high level logic programming language for robot control.
- Integration of reasoning, action and perception within a uniform theoretical and implementation framework.
- A prototype implementation of CONGOLOG.
- Fruitful connections with other areas of computer science (databases, software specification, programming languages, etc).
- Documentation:

  ftp://ftp.cs.toronto.edu:/pub/cogrob/README