The Situation Calculus and Golog

— A Tutorial —

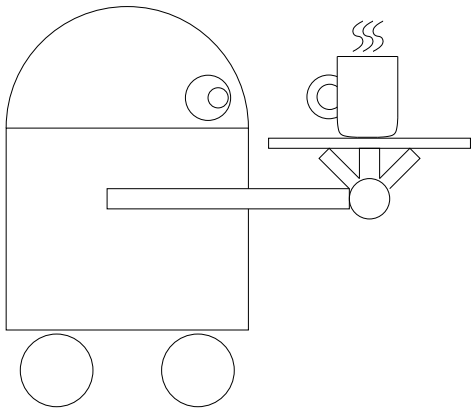
Gerhard Lakemeyer

Dept. of Computer Science
RWTH Aachen University
Germany

2nd Hybris Workshop, Freiburg, May 27–28, 2013

Outline

Knowledge-Based Agents



Example: An Office Robot

"If there are people in the office who want coffee, go to the kitchen, pick up coffee, and serve it to everyone who wants some. Alternatively, choose some room that is not clean and clean it. Alternatively, . . . "

Want a high-level representation that accounts for . . .

- incomplete knowledge
  - some room is dirty, but do not know which
  - agent gathers new knowledge at runtime (sensing)
- intelligent decision-making
  - programmer cannot foresee every eventuality
  - agent has to **reason about actions** (preconditions+effects)

Motivation: Cognitive Robotics

> *"Broadly speaking, the newly emerging field of cognitive robotics has, as its long term objectives, the provision of a uniform theoretical and implementation framework for autonomous robotic or software agents that reason, act and perceive in changing, incompletely known, unpredictable environments."*
>
> *[Ray Reiter,* Knowledge in Action*, 2001]*

The situation calculus is such a uniform framework.

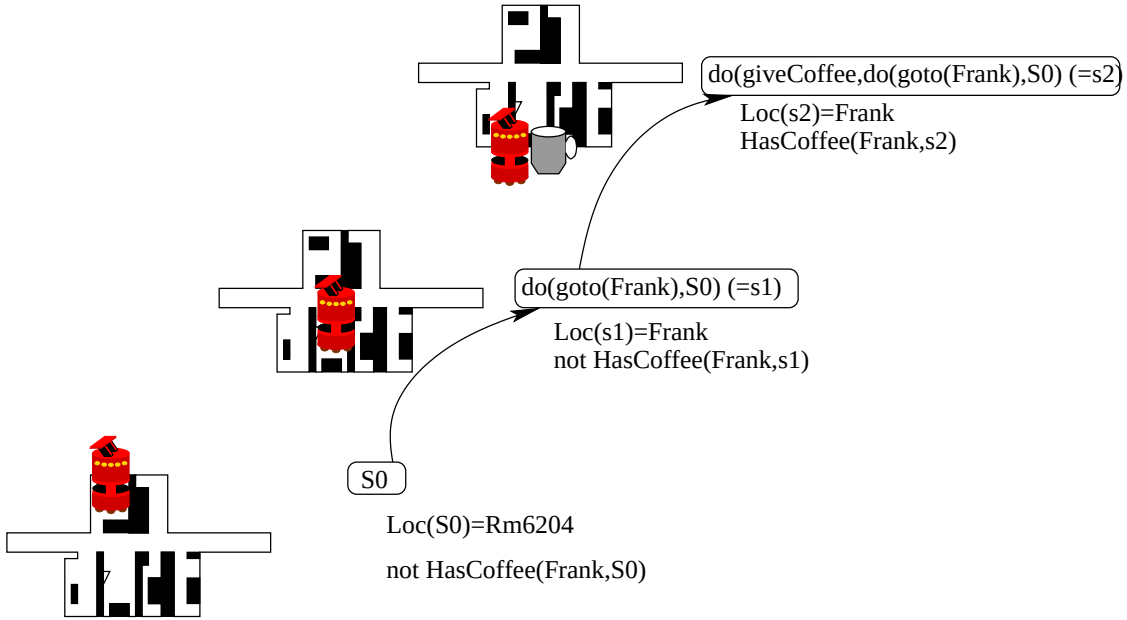
Goes back to John McCarthy (1963).

Here: variant developed by Reiter and his colleagues.

Literature

- Ray Reiter, *Knowledge in Action*, MIT Press 2001.
- Hector J. Levesque and Gerhard Lakemeyer, *Cognitive Robotics,* In F. van Harmelen, V. Lifschitz, and B. Porter, Eds., *Handbook of Knowledge Representation,* Elsevier, 2007.

The Situation Calculus in a Nutshell



- Situations are sequences of actions like $do(goto(Frank), S_0)$.
- Fluents characterizing situations: $\neg HasCoffee(Frank, S_0)$
- Axioms about change:
  $HasCoffee(p, do(a, s)) \equiv a = giveCoffee \vee HasCoffee(p, s)$

The Language (1)

The situation calculus is a sorted, second-order language with equality.

- There are three sorts: situations, actions, and ordinary objects.
- The logical connectives are $\neg, \wedge, \forall$.
  (We also use $\vee, \supset, \equiv, \exists$ as the usual abbreviations.)
- There are two function symbols of sort situation:
  - the constant S_0 denoting the initial situation
  - the function $do(a, s)$ denoting the situation obtained by doing action a in situation s
- A special binary predicate $s \sqsubset s'$ with the intended interpretation that s is a subsequence of s'.

The Language (2)

- A special binary predicate $Poss(a, s)$ with the intended interpretation that action a is executable in situation s.
- Relations and functions whose last argument is a situation are called fluents. E.g., in a world where objects can be painted $Color(x, c, s)$ may mean that x has the color c in situation s.

Notation: In the following, free variables are implicitly assumed to be universally quantified, e.g.,

$Poss(drop(r, x), s) \equiv Holding(r, x, s)$ stands for

$\forall r, s, x. Poss(drop(r, x), s) \equiv Holding(r, x, s)$.

Action Preconditions (Qualifications)

Actions have preconditions:

A robot r can lift object x in situation s iff r is not holding anything, is located next to x, and x is not too heavy:

$$Poss(pickup(r,x),s) \equiv \forall z.\neg Holding(r,z,s) \wedge \neg Heavy(x) \wedge NextTo(r,x,s).$$

r can repair an object iff the object is broken and he has glue:

$$Poss(repair(r,x),s) \equiv HasGlue(r,s) \wedge Broken(x,s)$$

Note: Defining $Poss$ this way makes a strong and simplifying assumption, ignoring other possible qualifications such as lifting an object is possible only if it is not glued to the ground, not slippery, etc. This is the qualification problem, which we simply ignore here.

Effect Axioms

Changes in the world are specified by effect axioms which describe how actions change fluents.

The effect on $Broken$ when r drops an object:

$$Fragile(x) \supset Broken(x, do(drop(r, x), s)).$$

(positive effect axiom)

Repairing an object causes the object not to be broken any more:

$$\neg Broken(x, do(repair(r, x), s)).$$

(negative effect axiom)

Frame Axioms

It takes more than effect axioms to describe a changing world. One needs so-called frame axioms to specify which fluents do not change when an action is executed.

- Positive frame axioms: e.g. dropping an object does not change its color:

  $$Color(x, c, s) \supset Color(x, c, do(drop(r, y), s)).$$

- Negative frame axioms: e.g. things do not break if they are not dropped:

  $$\neg Broken(x, s) \wedge [x \neq y \vee \neg Fragile(x)] \supset \neg Broken(x, do(drop(r, y), s)).$$

The Frame Problem

There are far too many frame axioms: only very few actions have an effect on any given fluent. All others remain unchanged.

- The color of an object does not change by lifting objects, opening doors, turning on lights, elect a president, etc.

There are about $2 \times \mathcal{A} \times \mathcal{F}$ frame axioms, where $\mathcal{A}$ is the number of actions and $\mathcal{F}$ the number of fluents.
Why is this a probem?

- Coding that many frame axioms by hand is awkward and error prone.
- It is very difficult to deal with that many axioms when reasoning (like planning a sequence of actions).

Goal: Find a compact representation of the frame axioms and derive them automatically from the effect axioms.

Reiter's Solution to the Frame Problem

Example: Suppose we have two positive effect axioms for $Broken$:

$$Fragile(x) \supset Broken(x, do(drop(r, x), s)),$$

$$NextTo(b, x, s) \supset Broken(x, do(explode(b), s)).$$

Then these can be transformed into an equivalent sentence:

$$[\exists r.(a = drop(r, x) \wedge Fragile(x)) \vee \exists b.(a = explode(b) \wedge NextTo(b, x, s))] \supset Broken(x, do(a, s)). \quad (1)$$

Similarly, the negative effect axiom

$$\neg Broken(x, do(repair(r, y), s))$$

can be rewritten as:

$$\exists r.(a = repair(r, x)) \supset \neg Broken(x, do(a, s)) \quad (2)$$

A Completeness Assumption

Now we make the following completeness assumption:

Axiom (1) characterizes all possible conditions when an action a can break an object x.
That means: if $\neg Broken(x, s)$ and $Broken(x, do(a, s))$ are true, then the truth value of $Broken$ can only have changed because

$$\exists r.(a = drop(r, x) \wedge Fragile(x)) \qquad \text{or}$$
$$\exists b.(a = explode(b) \wedge NextTo(b, x, s))$$

were true.
Similarly, we assume that Axiom (2) characterizes the only possibility when an action a can make an object x whole.

Successor State Axioms

The effect axioms together with the completeness assumption can be combined in a so-called successor state axiom SSA_{Br} for $Broken$:

$$Broken(x, do(a, s)) \equiv$$
$$\exists r.(a = drop(r, x) \wedge Fragile(x)) \vee$$
$$\exists b.(a = explode(b) \wedge NextTo(b, x, s)) \vee$$
$$(Broken(x, s) \wedge \neg(\exists r.a = repair(r, x))).$$

From the SSA one can deduce all effect axioms and all frame axioms: Examples:

$$\text{SSA}_{Br} \models Fragile(x) \supset Broken(x, do(drop(r, x), s))$$

$$\text{SSA}_{Br} \wedge \text{UNA} \models \neg Broken(x, s) \wedge \neg Fragile(x) \supset \neg Broken(x, do(drop(r, y), s)),$$

where UNA is $drop(r, x) \neq explode(b)$

The General Case

The previous example can be generalized. Suppose we have the following normal form for effect axioms:
One positive effect axiom for each fluent F:

$$\gamma_F^+(\vec{x}, a, s) \supset F(\vec{x}, do(a, s)).$$

One negative effect axiom for each fluent F:

$$\gamma_F^-(\vec{x}, a, s) \supset \neg F(\vec{x}, do(a, s)).$$

Here $\gamma_F^+(\vec{x}, a, s)$ and $\gamma_F^-(\vec{x}, a, s)$ are wffs whose free variables are $\vec{x}$, a, and s. Such a normal form can always be obtained assuming we have unique names axioms for actions.
Then the SSA has the following form:

$$F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee [F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s)]$$

Solution to the Frame Problem (Summary)

Our solution yields the following axioms:

- For each fluent F there is a single SSA:
  $F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s))$
- For each action A there is a single precondition axiom:
  $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$.
- Unique names axioms for all actions.

Ignoring unique names axioms, we have $\mathcal{F} + \mathcal{A}$ axioms, compared to $2 \times \mathcal{F} \times \mathcal{A}$ frame axioms!

Note: SSAs are fairly short in real applications.

- The length of a SSA is roughly proportional to the number of actions which have an effect on the fluent.
- In practice only very few actions have an effect on a particular fluent.

Basic Action Theories

Action Preconditions: Σ_{pre}

$$Poss(goto(x), s) \equiv true$$
$$Poss(loadCoffee, s) \equiv \neg HasCoffee(R, s)$$
$$Poss(giveCoffee(p), s) \equiv HasCoffee(R, s)$$

Successor State Axioms for Fluents: Σ_{post}

$$HasCoffee(p, do(a, s)) \equiv$$
$$[p = R \wedge a = loadCoffee] \vee [a = giveCoffee(p)] \vee$$
$$[HasCoffee(p, s) \wedge \neg(\exists p'.p = R \wedge a = giveCoffee(p'))]$$

$$Loc(do(a, s)) = x \equiv$$
$$[a = goto(x)] \vee [Loc(s) = x \wedge \neg(\exists y.y \neq x \wedge a = goto(y))]$$

What is true initially: Σ_0

$$Loc(S_0) = Rm6304, \neg HasCoffee(Frank, S_0), HasCoffee(R, S_0).$$

$\Sigma = \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup$ UNA $\cup$ found. axioms

$$\Sigma \models HasCoffee(Frank, do(giveCoffee(Frank, goto(Frank, S_0)))).$$

Foundational Axioms

Besides domain dependent axioms to characterize actions, we need a few domain independent axioms which specify, roughly, that situations are all and the only sequences of actions starting at S_0.

1. Situations have unique names:
   $do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2$
2. There are no situations other than those reachable from S_0:
   $\forall P.P(S_0) \wedge \forall a, s.[P(s) \supset P(do(a, s))] \supset \forall s.P(s)$
3. The definition of $\sqsubset$ as subhistory:
   $\neg s \sqsubset S_0$
   $s \sqsubset do(a, s') \equiv s \sqsubseteq s'$
   where $s \sqsubseteq s'$ is an abbreviation for $s \sqsubset s' \vee s = s'$.

Reasoning about Action by Regression

Let us consider the subproblem $\Sigma \models G(\sigma)$,

where $\sigma = do([\alpha_1, \ldots, \alpha_n], S_0)$ is the only situation mentioned in G (regressable formula).

$(do([\alpha_1, \alpha_2], s) \stackrel{def}{=} do(\alpha_2, do(\alpha_1, s)))$

Idea: Replace each fluent $F(\vec{t}, \sigma)$ with $F(\vec{x}, do(a, s)) \equiv \Phi(\vec{x}, a, s)$ by $\Phi(\vec{t}, \alpha_n, do([\alpha_1, \ldots, \alpha_{n-1}], S_0))$,

and continue until S_0 is the only situation left.

Simlarly, replace $Poss(A, s)$ by its definition.

Then we only need to reason about the initial situation.

An Example

$$G = Poss(giveCoffee(F), S_0) \wedge HasCoffee(F, do(giveCoffee(F), S_0)).$$

$$\begin{aligned} &Poss(giveCoffee(p), s) \equiv HasCoffee(R, s) \\ &HasCoffee(p, do(a, s)) \equiv \\ &\quad [p = R \wedge a = loadCoffee] \ \vee [a = giveCoffee(p)] \ \vee \\ &\quad [HasCoffee(p, s) \wedge \neg(\exists p'.p = R \wedge a = giveCoffee(p'))] \end{aligned}$$

Let $\mathcal{R}[G]$ denote the formula obtained by applying regression.

Then $\mathcal{R}[G]$ = $HasCoffee(R, s) \wedge$
$[F = R \ \wedge \ giveCoffee(F) = loadCoffee] \ \vee$
$[giveCoffee(F) = giveCoffee(F)] \vee$
$[HasCoffee(F, S_0) \wedge \neg(\exists p'.F = R \wedge giveCoffee(F) = giveCoffee(p'))]$

which simplifies to $\mathcal{R}[G] = HasCoffee(R, S_0)$.

Note, in particular, that $\Sigma_0 \cup$ UNA $\models \mathcal{R}[G]$.

This is no coincidence!

Main Result about Regression

Let G be a regressable sentence and $\mathcal{D}$ a basic action theory. Then

1. $\mathcal{R}[G]$ mentions only S_0.
2. $\Sigma \models G \equiv \mathcal{R}[G]$.
3. $\Sigma \models G$ iff $\Sigma_0 \cup \text{UNA} \models \mathcal{R}[G]$.

Note: (3) says that, after applying the regression operator, one only needs to consider Σ_0 and, most importantly, figuring out whether G follows from Σ reduces to ordinary first-order theorem proving!

Planning in the Situation Calculus

Let $\sigma = do([a_1, \ldots, a_n], S_0)$ and $executable(\sigma)$ stand for a formula expressing that each a_i is executable.

For a given basic action theory Σ and goal G planning means finding a situation σ such that

$$\Sigma \models executable(\sigma) \wedge G(\sigma).$$

For a particular σ use regression to compute the entailment.

Planning is too Expensive

- While basic action theories allow to infer arbitrary plans in principle, this often does not work in practice.
- Even optimized planners fail in general as this is an inherently hard problem.
- Perhaps not even needed: Often the user has a good idea of what needs to be done.

Idea: Tell the robot what to do if possible.
Leave (small) subtasks for planning where needed.

Coffee delivery robot

Imagine a robot serving coffee to many people.
The robot can carry only one cup at a time.

Golog

proc $DeliverCoffee$
  while $\exists x.WantsCoffee(x)$ **do**
    $\pi x.WantsCoffee(x); ServeCoffee(x)$
  endWhile
  $goto(Home)$
endProc

proc $ServeCoffee$(x)
  if $\neg HasCoffee(R)$ **then** $goto(CoffeeM); loadCoffee$ **endIf**
  $goto(x); giveCoffee(x)$
endProc

- Semantics of programs given in the situation calculus:
  $Do(\rho, s, s')$: executing program ρ leads from situation s to s'.
- Program execution: $\Sigma \models \exists s.Do(DeliverCoffee, S_0, s)$
  (finds as a side effect a sequence of executable actions)

- Implementation in Prolog.
- Efficient if the initial state consists of a set of literals.

Complex Actions

Programming in the situation calculus is introduced by defining the following complex actions:

- Sequence:
  First execute A, then B.
- Test actions:
  Is ϕ true in the current situation?
- If-then-else, While-loops
- Nondeterministic actions:
  Execute A or B.
- Nondeterministic choice of the argument of an actions:
  e.g. $(\pi x)pickup(x)$: pick up an arbitrary object.
- procedures
  Names for complex actions, parameters, recursion.

The Meaning of Complex Actions (1)

$Do(a, s, s')$ stands for:
The execution of action a in situation s leads to situation s'.

- Primitive actions:

  $$Do(a, s, s') \doteq Poss(a[s], s) \wedge s' = do(a[s], s).$$

- Sequence:

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

- Nondeterministic actions:

  $$Do([a_1|a_2], s, s') \doteq Do(a_1, s, s') \vee Do(a_2, s, s')$$

- Nondeterministic choice:

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

- Test actions:

  $$Do(\phi?, s, s') \doteq \phi[s] \wedge s = s'$$

The Meaning of Complex Actions (2)

- If-then-else:

$$Do(if(\phi, a_1, a_2), s, s') \doteq Do([[\phi?; a_1]|[\neg\phi?; a_2]], s, s')$$

- *-operator: execute the action n times $(n \geq 0)$:

$$Do(a^*, s, s') \doteq \forall P[\forall s_1 P(s_1, s_1) \wedge \forall s_1, s_2, s_3 (P(s_1, s_2) \wedge Do(a, s_2, s_3) \supset P(s_1, s_3))] \supset P(s, s')$$

- While-loops:

$$Do(while(\phi, a), s, s') \doteq Do(([\phi?; a]^*; \neg\phi?), s, s')$$

- Procedures:

  definition is second order because of recursion (omitted).

Note: The semantics of complex actions is completely described within the situation calculus.

Example

Consider the coffee-delivery program together with the basic action theory as before and the initial state

$$\Sigma_0 = \{\forall x.WantsCoffee(x) \equiv (x = Frank), \neg HasCoffee(R, S_0)\}.$$

$$\Sigma \models Do(DeliverCoffee, S_0, \sigma),$$

where $\sigma = do([goto(CoffeeM), loadCoffee,$
$goto(Frank), giveCoffee(Frank), goto(Home)], S_0).$

Golog: alGOl in LOGic

Comparison to conventional imperative programming:

- Procedures in “normal” imperative languages are ultimately reduced to machine instructions like assignment statements.
- Procedures in Golog actions are reduced to primitive actions which refer to actions in the real world, like picking up objects or moving. Instead of machine states we are interested in world states.
- Executing complex actions requires logical reasoning about what the world is like and how it evolves. For example, in order to execute
  while $[\exists x.WantsCoffee(x)]$ do ...
  one needs to evaluate the test $[\exists x.WantsCoffee(x)]$ in the current situation.

Extensions of the Basic Framework

Over the past years a number of extensions to Golog have been proposed to make it applicable in realistic domains, especially in robotics. These include

- concurrent actions
- time
- continuous change
- an explicit notion of knowledge
- sensing
- stochastic actions