Summary of "Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems" (2001)
Book's Author: Raymond Reiter

** Obviously, I don't take any credit for the material in this file. This is just for my own reference.*

Chapter 6: Complex Actions, Procedures, and Golog

Intuitively, $Do(\delta, s, s')$ says that *it is possible to reach situation s' from situation s by executing a sequence of actions specified by δ.*
Note that complex actions may be nondeterministic, that is, may have several different executions terminating in different situations.

Do is defined inductively on the structure of its first argument as follows:

- Primitive actions:
  $Do(a, s, s') \stackrel{def}{=} Poss(a[s]^1, s) \wedge s' = do(a[s], s)$.
- Test actions:
  $Do(\phi?, s, s') \stackrel{def}{=} \phi[s]^2 \wedge s' = s$.
- Sequence:
  $Do(\delta_1; \delta_2, s, s') \stackrel{def}{=} (\exists s'').Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$.
- Nondeterministic choice of two actions:
  $Do(\delta_1 | \delta_2, s, s') \stackrel{def}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$.
- Nondeterministic choice of action argument:
  $Do((\pi x)\delta(x), s, s') \stackrel{def}{=} (\exists x) Do(\delta(x), s, s')$.
- Nondeterministic iteration: Execute δ zero or more times.

$$Do(\delta^*, s, s') \stackrel{def}{=}$$
$$(\forall P).\{(\forall s_1)P(s_1, s_1) \wedge (\forall s_1, s_2, s_3)[Do(\delta, s_1, s_2) \wedge P(s_2, s_3) \supset P(s_1, s_3)]\} \supset P(s, s').$$

Conditionals and while-loops can be defined in terms of the previous constructs as follows:

$$\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf} \stackrel{def}{=} [\phi?; \delta_1] | [\neg\phi?; \delta_2]$$
$$\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile} \stackrel{def}{=} [\phi?; \delta]^*; \neg\phi?$$

[1] The notation $a[s]$ means the result of restoring the situation argument s to all functional fluents mentioned by the action term a. For example, if a is $goTo(location(Sam))$, and if *location* is a functional fluent, then $a[s]$ is $goTo(location(Sam, s))$.

[2] Here, ϕ is a situation-suppressed expression (not a situation calculus formula) consisting of a formula in the language of the situation calculus, but with all situation arguments suppressed. $\phi[s]$ denotes the situation calculus formula obtained from ϕ by restoring situation variable s into all fluent names (relational and functional) mentioned in ϕ.

Chapter 7: Time, Concurrency, and Processes[3]

Insofar as actions are currently represented in the situation calculus, they occur sequentially, and atemporally. But real actions (1) normally have **duration**s, (2) they occur in **time**, and (3) frequently they occur together, i.e., **concurrent**ly.

Modelling Actions with duration
- For each action with duration, introduce two instantaneous actions *start* and *end* and a relational fluent. For example for the action *press*, define *startPress* and *endPress* and the fluent $pressing(s)$.
- The successor state axiom for the fluent will be:

$$pressing(do(a, s)) \equiv a = startPress \vee (pressing(s) \wedge a \neq endPress)$$

Modelling Time (explicitly)
The situation calculus axioms capture a purely qualitative notion of time. Sequential action occurrence is the only temporal concept captured by the axioms; an action occurs *before* or *after* another. It may occur one millisecond or one year before its successor; the axioms are neutral on this question. So a situation $do([A_1, A_2, \ldots, A_n], S_0)$ must be understood as a world history in which, after a nondeterministic period of time, A_1 occurs, then, after a nondeterministic period of time, A_2 occurs, etc.

- Add a new temporal argument to all instantaneous actions, denoting the actual time at which that action occurs. So we have actions $A(\vec{x}, t)$

- Define a new function symbol $time : action \rightarrow reals$[4] . $time(a)$ denotes the time of occurrence of action a. This means that in any application involving a particular action $A(\vec{x}, t)$, we shall need an axiom specifying the occurrence time of the action A: $time(A(\vec{x}, t)) = t$.

- It is convinient to have a new function symbol $start : situation \rightarrow reals$.
$start(s)$ denotes the start time of situation s. This requires the new foundational axiom: (defining the start time of S_0 is arbitrary).

$$(\forall s, a)\ start(do(a, s)) = time(a)$$

- We may also define a new function symbol $end : situation \times action \rightarrow reals$.
$end(s, a)$ denotes the ending time of situation s which is marked by occurance of action a. We have

$$(\forall s, a)\ end(s, a) = start(do(a, s))$$

- We need to modify the definition of *executable* by ruling out the possibility of prior actions having later times:

$$executable(s) \stackrel{def}{=} (\forall a, s').do(a, s') \sqsubseteq s \ \supset\ Poss(a, s') \wedge start(s') \leq time(a).$$

This ensures that $(\forall a, s)\ start(s) \leq end(s, a)$

For Golog semantics, we just need to change the definition of the *Do* macro for primitive actions:

$$Do(a, s, s') \stackrel{def}{=} Poss(a[s], s)\ \wedge\ start(s) \leq time(a[s])\ \wedge\ s' = do(a[s], s)$$

Modelling Concurrency
There are two types of concurrency:

- *Interleaved Concurrency*: Two actions are interleaved when one of them is the next action to occur after the other

[3]Part of the content is from "J. Pinto, R. Reiter, *Time in the Situation Calculus.* 1995".
[4]we could use integers, rationals, or anything else on which a binary relation $<$ is defined.

- *True Concurrency*: When two or more actions start and end at exactly the same times.

In computer science, concurrency is most often modeled via interleaving. Interleaved concurrency can be expressed within the sequential situation calculus, provided we appeal to instantaneous start and end actions, as defined above. We define true concurrency in two stages; first a non-temporal account and then a temporal account.

The Concurrent, Non-Temporal SitCalc

- Concurrent actions are sets of simple actions, we can use the notation $a \in c$ to mean that simple action a is one of the actions of the concurrent action c.

- To distinguish the sorts *action* of simple actions and *concurrent*, we use variables $a, a', \ldots,$ and $c, c', \ldots,$ respectively.

- We extend the function symbol do to take concurrent actions as an argument. Then we have situation terms like $do(\{startMeeting(Sue), collide(A, B)\}, S_0)$.

- We need to simply replace each *action* variable in the foundational axioms by a variable of sort *concurrent action*.

- The abbreviation for $executable(s)$ becomes:
$executable(s) \stackrel{def}{=} (\forall c, s').do(c, s') \sqsubseteq s \supset Poss(c, s')$

- We need to add two more axioms to complete the foundational axioms of the concurrent, non-temporal SitCalc:
$Poss(a, s) \supset Poss(\{a\}, s)$,
$Poss(c, s) \supset (\exists a)a \in c \wedge (\forall a)[a \in c \supset Poss(a, s)]$
(the converse need not hold because of the *precondition interaction pronlem*)

The Concurrent, Temporal SitCalc

- A concurrent action makes no intuitive sense if it is empty, or if it contains two or more simple actions whose occurrence times are different. Accordingly, define the notion of a *coherent* concurrent action to be one for which there is at least one action in the collection, and for which all of the (instantaneous) actions in the collection occur at the same time:
$coherent(c) \stackrel{def}{=} (\exists a)a \in c \wedge (\exists t)(\forall a')[a' \in c \supset time(a') = t]$

- Extend the function $time$ from simple actions to concurrent ones:
$coherent(c) \supset [time(c) = t \equiv (\exists a)(a \in c \wedge time(a) = t)]$

- $start(do(c, s)) = time(c)$

- $executable(s) \stackrel{def}{=} (\forall c, s').do(c, s') \sqsubseteq s \supset Poss(c, s') \wedge start(s') \leq time(c)$.

- We keep all the foundational axioms of the concurrent, non-temporal SitCalc, we just need to generalize the last one: $Poss(c, s) \supset coherent(c) \wedge (\forall a)[a \in c \supset Poss(a, s)]$

- For Golog semantics, we need only change the definition of the Do macro for sequential, temporal Golog to apply to concurrent actions instead of simple actions:
$Do(c, s, s') \stackrel{def}{=} Poss(c[s], s) \wedge start(s) \leq time(c[s]) \wedge s' = do(c[s], s)$

Chapter 11: Sensing and Knowledge[5]

- $K(s', s)$ denotes "s' is accessible from s".

- $\mathbf{Knows}(\phi, s) \stackrel{def}{=} (\forall s').K(s', s) \supset \phi[s']$
Where ϕ is a situation suppressed expression. $\phi[s]$ denotes the formula obtained from ϕ by restoring situation variable s into all fluent names mentioned in ϕ.

- $\mathbf{KWhether}(\phi, s) \stackrel{def}{=} \mathbf{Knows}(\phi, s) \vee \mathbf{Knows}(\neg\phi, s)$

- $\mathbf{KRef}(t, s) \stackrel{def}{=} (\exists x)\mathbf{Knows}(x = t, s)$ (t is a term that does not mention x)

-There are two kinds of actions:

- Ordinary/Physical actions that change the state of the world
- *Knowledge-producing actions* that change the state of the knowledge of an agent

- Knowledge-producing actions can be divided into two groups:

- Actions of *sense* type, which make known the truth value of a relational fluent. For each relational fluent, there is a *sense* action. We have that $\mathbf{KWhether}(P, do(sense_P, s))$ holds. Which means:
  $(\forall \vec{x}, s).\phi(\vec{x}, s) \supset \mathbf{Knows}(\phi(\vec{x}), do(sense_\phi(\vec{x}), s))$,
  $(\forall \vec{x}, s).\neg\phi(\vec{x}, s) \supset \mathbf{Knows}(\neg\phi(\vec{x}), do(sense_\phi(\vec{x}), s))$.
- Actions of *read* type, which make known the denotation of a functional fluent. For each functional fluent, there is a *read* action. We have that $\mathbf{KRef}(t, do(read_t, s))$ holds.
  $(\forall \vec{x}, s)\mathbf{KRef}(f(\vec{x}), do(read_f(\vec{x}), s))$.

- Successor state axiom for K:
$K(s'', do(a, s)) \equiv (\exists s').s'' = do(a, s') \wedge K(s', s) \wedge Poss(a, s') \wedge \text{SR}(a, s) = \text{SR}(a, s')$

Where SR is called *sensing result function* and is defined as follows:

$\text{SR}(sense_Q, s) = r \equiv (r = \text{"Yes"} \wedge Q(s)) \vee (r = \text{"No"} \wedge \neg Q(s))$
$\text{SR}(read_t, s) = r \equiv r = t(s)$
For ordinary actions, the result is always the same, with the specific result not being significant. So for an ordinary action a, we could have
$\text{SR}(a, s) = r \equiv r = \text{"ok"}$

- For simplicity we assume that all actions are to be axiomatized as affecting only either the K fluent or other fluents (Knowldge-producing actions only affect the K fluent, this is called the *no-side-effect assumption*). This ensures a sharp division between knowledge-producing actions and ordinary actions. Without this policy there is nothing to prevent us from having an action such as *OpenBag* which causes the bag to be open and makes the agent aware of the content of the bag. But this policy does not restrict the capabilities of the agents that we model as we can always follow an open action (which only causes the bag to be open) by a sense action (which causes the agent to know what the contents of the bag are).

- Knowledge-producing actions do not change the state of the world. The only fluent whose truth value is altered by a knowledge-producing action is K.

[5]Part of the content is from "R. B. Scherl and H. J. Levesque. *Knowledge, Action, and the Frame Problem.* Artificial Intelligence, 144(1-2):139, 2003."

Theorem 1. Suppose a is a knowledge-producing action. Then $\forall s \forall P(\neq K)$ if $P(s)$ then $P(do(a,s))$.

- Actions only affect knowledge in the appropriate way. There are no unwanted increases in knowledge.
Theorem 2. Let a be an action that doesn't affect fluent P (i.e. $\forall s P(s) \equiv P(do(a,s))$. Then

$$\underbrace{(\exists s').K(s',s) \wedge \neg P[s']}_{\neg \mathbf{Knows}(P,s)} \wedge Poss(a,s') \wedge \text{SR}(a,s) = \text{SR}(a,s') \qquad \text{iff} \qquad \neg \mathbf{Knows}(P, do(a,s))$$

- Agents never forget. Informally speaking, if the agent knows P at s, then P is also known at $do(a,s)$ as long as the effect of a is not to make P false.
Theorem 3. $\forall P, s$ if $\mathbf{Knows}(P,s)$ holds then $\mathbf{Knows}(P, do(a,s))$ holds as long as the axiomatization entails $(\forall s).P(s) \equiv P(do(a,s))$.

- The notation $\mathbf{Knows}(a,s)$ suffers from one limitation: ϕ must be a situation-suppressed expression, and therefore, its situation restored version $\phi[s]$ must be uniform in s. Therefore, the **Knows** notation can only express knowledge about uniform formulas. Accordingly, we expand the class of expressions ϕ by introducing a special symbol now, and we allow $\phi(now)$ to be any situation-suppressed expression that may also mention the special term now:

$\mathbf{Knows}(\phi(now), s) \stackrel{def}{=} (\forall s').K(s',s) \supset \phi(s')$

E.g. $\mathbf{Knows}((\exists s^*) now = do(a,s^*), do(a,s))$ expands to $(\forall s').K(s', do(a,s)) \supset (\exists s^*) s' = do(a,s^*)$.

New Foundational Axioms for Situations
The following five axioms form the new set of foundational axioms. We have the following axioms from before:

$$do(a_1,s_1) = do(a_2,s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \tag{1}$$
$$\neg s \sqsubset S_0, \tag{2}$$
$$s \sqsubset do(a,s') \equiv s \sqsubset s'. \tag{3}$$

- The sort $Init$ contains S_0 and the situations which are K-accessible from an $Init$ situation. Nothing else is in $Init$:
$Init(s) \stackrel{def}{=} \neg(\exists a, s') s = do(a,s')$.
we need a weaker version of the old induction axiom:

$$(\forall P).(\forall s)[Init(s) \supset P(s)] \wedge (\forall a, s)[P(s) \supset P(do(a,s))] \supset (\forall s) P(s). \tag{4}$$

Finally, insist that only initial situations can be K-related to an initial situation:

$$K(s,s') \supset [Init(s) \equiv Init(s')] \tag{5}$$

- To provide inductive proofs of properties that are not about the K fluent, we must now prove $(\forall s).S_0 \sqsubset s \supset \phi(s)$ instead of proving $(\forall s)\phi(s)$.

- provided any one of the reflexive, transitive, symmetry and Euclidean properties on accessibility relation K is true in all initial situations, then it will be true in all situations.

Some Definitions

The **planning** problem
Given an axiomatized initial situation, and a goal statement, find an action sequence that will lead to a state in which the goal will be true.

The **ramification** problem
Solving the frame problem in presence of state constraints as initial data.

The **qualification** problem
Determining action preconditions when given a set of qualifications for an action together with a set of state constraints.

The **projection** problem
Given a sequence of ground action terms, and a formula G, determine whether G is true in the situation resulting from performing these actions. In other words, we are interested in answering queries of the form: Would G be true in the world resulting from the performance of the given sequence of actions?

Vahid Vaezian (vvaezian [at] sfu.ca), February 3, 2018