Knowledge, Action, and Ability in the Situation Calculus

Hector Levesque
Dept. of Computer Science
University of Toronto
and
Canadian Institute for Advanced Research

Joint work with

Yves Lespérance
Fangzhen Lin
Richard Scherl

Cognitive Robotics

Principled accounts of the relationships among knowledge, perception, and action

Toronto group: co-authors + Ray Reiter + students

Theoretical framework formulated in the language of the situation calculus

- exploiting a recent solution to the frame problem
- with extensions to handle (eventually)
  - complex actions
  - knowledge
  - state constraints
  - ☞ ability
  - indexicals
  - exogenous events, multiple agents
  - noisy sensors / effectors
  - ...

Results in **GOLOG**, a novel programming language for high-level robotic control

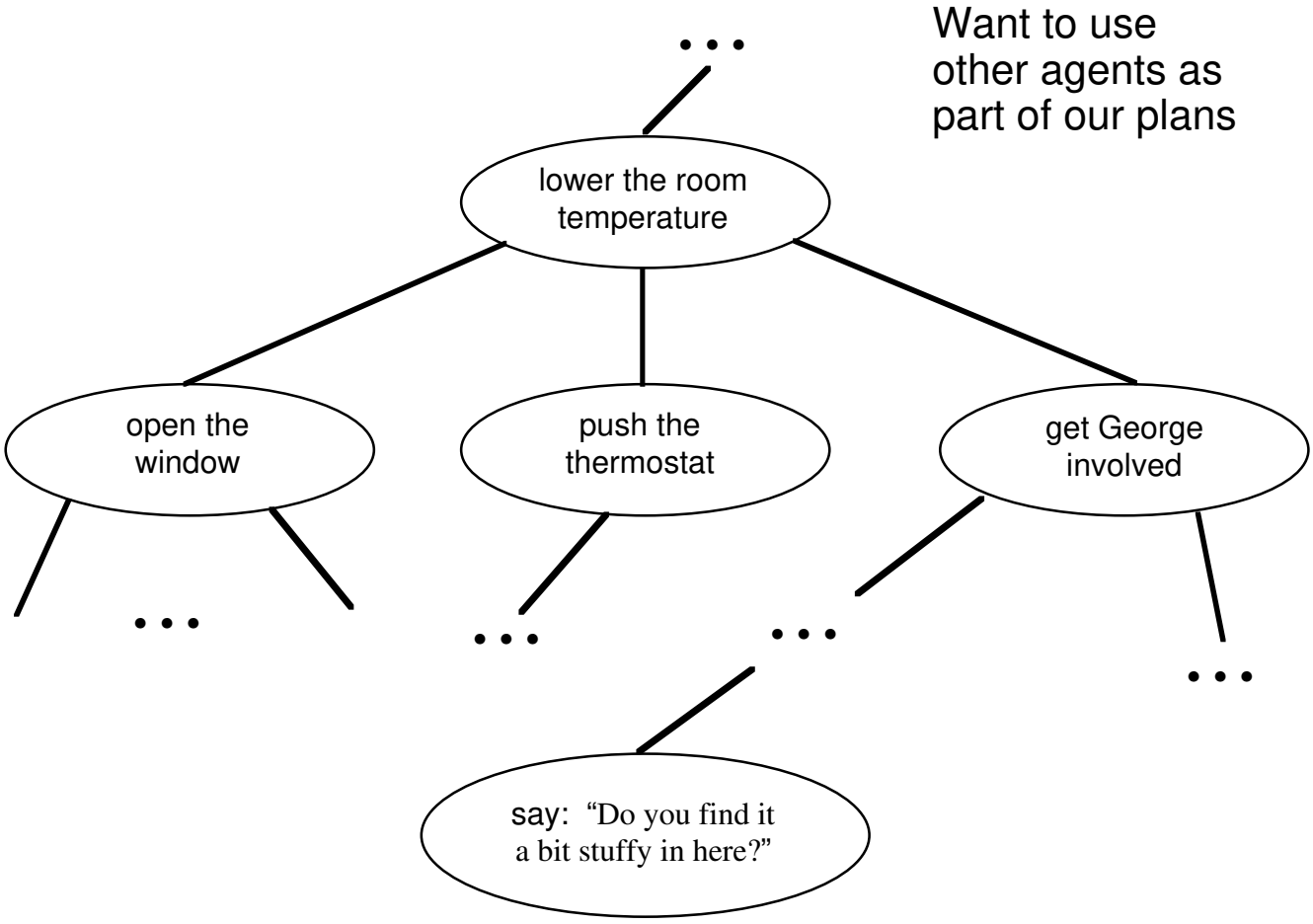
- prototype implementation is running
- currently: link to simulated actions, sensors

  "Truckworld" variant
- anticipated: link to ARK mobile platform

Introduction

Why communicate?

- to indicate our mental state? ✘
- to affect the mental state of others? ✘
- to advance our non-linguistic goals! ✔



But then: What do I need to *know* about the agent, and what do I need to *do* to use the agent in this way?

Delegation

Understand agents as acting *rationally*, given a set of beliefs and goals (which vary over time)

The Principle

> If there is an agent who
>
> 1. knows how to bring about ϕ
> 2. is committed to doing so
>
> then ϕ will obtain

This principle provides a strategy for achieving ϕ, by getting another agent into a certain mental state

and leaving the rest to him/her/it

But what is this mental state?

- committed to the goal of achieving ϕ

  see Cohen and Levesque: intention

☞ • knowing how to achieve ϕ: ability

  would like an account that is sufficient to validate above principle

  but here: single agent case only

Planning?

If it turns out that

knowing how to achieve ϕ

$\equiv$

having a plan P for achieving ϕ

why not simply use a formalization of planning?

The issue is not

planning = *finding* a plan

but

ability = *having* a plan

For example, need to be able to reason from facts like

Bill knows how to open the mystery lock.

without *our* being able to generate a plan to do it.

Need a specification of $Can(\phi)$ that is more than just a procedure for generating plans for ϕ.

planning procedure should be correct
relative to this specification

Overview

1. Motivation
2. This slide
3. Crash course in the situation calculus:
   - the frame problem
   - complex actions
   - knowledge
4. Achieving a goal
   - the classical approach
   - using programs
   - iterative / fixed point versions
5. A path-based definition of CAN
   - some properties
   - relation to programs
6. Rethinking the “what” of planning

Advisory: The following presentation contains explicit formulas and other highly graphic material.

Situation Calculus

McCarthy

A second-order logical language for representing dynamically changing worlds

- The world is thought of as being in some state
- This state changes only as the result of an agent performing some named action

Syntactically: predicate calculus with special sorts for

- actions: a, jump, rotate(90°), putIn(b, box3)
- situations: s, S_0, $do(a, s)$ ← the state resulting from performing action a in state s

  $s_1 \leq s_2$

Fluents: predicates (or functions) whose final argument is a situation

$$\text{Happy}(S_0), \quad \text{NextTo}(x, y, s), \quad \text{location}(x, do(\text{kick}(x), s))$$

Actions may have preconditions: $Poss(a, s)$

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

World dynamics are specified by <u>effect axioms</u> like

$$Poss(\text{drop}(r,x), s) \wedge \text{Fragile}(x) \supset \text{Broken}(x, do(\text{drop}(r,x), s)) \quad \text{+ve}$$

$$Poss(\text{repair}(r,x), s) \supset \neg\text{Broken}(x, do(\text{repair}(r,x), s)) \quad \text{–ve}$$

The Frame Problem

McCarthy & Hayes

Given a specification of the "causal laws" via effect axioms, derive a succinct specification of what does *not* change as the result of performing an action

Should not have to write down frame axioms like

$$Poss(\text{drop}(r,y), s) \wedge \neg\text{Broken}(x, s) \wedge (x \neq y \vee \neg\text{Fragile}(x)) \supset \neg\text{Broken}(x, do(\text{drop}(r,y), s))$$

Solution: Pednault, Haas, Schubert, Reiter

- Can group all the effect axioms for a fluent F into two:

$$Poss(a, s) \wedge \gamma^{+}(\bar{x}, a, s) \supset F(\bar{x}, do(a,s))$$
$$Poss(a, s) \wedge \gamma^{-}(\bar{x}, a, s) \supset \neg F(\bar{x}, do(a,s))$$

- Assume these axioms characterize all the conditions under which any action a can change F

$$Poss(a, s) \wedge \neg F(\bar{x}, s) \wedge F(\bar{x}, do(a,s)) \supset \gamma^{+}(\bar{x}, a, s)$$
$$Poss(a, s) \wedge F(\bar{x}, s) \wedge \neg F(\bar{x}, do(a,s)) \supset \gamma^{-}(\bar{x}, a, s)$$

- Under suitable conditions, these four together are logically equivalent to a single successor state axiom

$$Poss(a, s) \supset [F(\bar{x}, do(a,s)) \equiv \gamma^{+}(\bar{x}, a, s) \vee F(\bar{x}, s) \wedge \neg\gamma^{-}(\bar{x}, a, s)]$$

For example: $Poss(a, s) \supset [\text{Broken}(x, do(a,s)) \equiv \exists r \{a = \text{drop}(r,x) \wedge \text{Fragile}(x)\} \vee \text{Broken}(x, s) \wedge \neg\exists r\, a = \text{repair}(r,x)$

The Frame Problem (contd.)

For each fluent F, the successor state axiom for F reduces $F(do(a,s))$ to some condition on the state s

$$Poss(a,s) \supset F(do(a,s)) \equiv \Phi(s)$$

this entails appropriate frame axioms

- significantly generalizes STRIPS

  wrt initial state S_0 and effect axioms

The conciseness of the succ. state axiom depends on

- quantification over actions
- assumption that relatively few actions will affect a given fluent
- assumption that <u>all</u> effect axioms have been enumerated and are of the form

$$Poss(-, s) \wedge \ldots \supset \pm F(do(-, s))$$

Limitations:

- only actions with determinate effects

  ✘ $Poss(\text{flip}, s) \supset \text{Heads}(do(\text{flip}, s)) \vee \text{Tails}(do(\text{flip}, s))$

- actions are all independent primitives

  no sequences, iterations, *etc.*

  see below

- no indirect effects

  all effect axioms must be in hand

  (ramification problem)

Complex Actions in the Situation Calculus

Would like a way of expressing and reasoning about actions that are constructed out of primitive actions

- **`If`** Car_In_Driveway **`then`** drive **`else`** walk
- **`While`** Tree_Still_Standing **`do`** chop
- $\Pi\, x$ [Block(x)?; grasp(x)]

Need to extend the notation $do(a, s) = s'$ to handle a complex action / program A:

$Do(A, s, s')$

s' is one possible final situation that could result from doing program A in situation s

However, previous solution to the frame problem depended on all actions being determinate primitives

(and would like to inherit this solution)

Method: treat $Do(...)$ as a macro that expands to a formula in the sit. calculus that only uses $do(...)$

Result: complex actions are completely eliminated by macro expansion

Limitation: cannot quantify over complex actions

- can only use in the context $Do(A, s, s')$, where A is a specific program

The Do Macro

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

2. $Do([A;B], s, s') \doteq \exists s^* Do(A, s, s^*) \wedge Do(B, s^*, s')$

3. $Do([\textbf{if } \phi \textbf{ then } A \textbf{ else } B], s, s') \doteq$
$(\phi[s] \supset Do(A, s, s')) \wedge (\neg\phi[s] \supset Do(B, s, s'))$

4. $Do([\textbf{while } \phi \textbf{ do } A], s, s') \doteq$
$\forall P (\bullet\bullet\bullet) \supset P(s , s')$ 2nd order!

... others

$\forall s_1 \neg\phi[s_1] \supset P(s_1, s_1)$

$\forall s_1 s_2 s_3 \ \phi[s_1] \wedge Do(A, s_1, s_2) \wedge P(s_2,s_3) \supset P(s_1, s_3)$

Notation

$\phi[s]$ means take all fluents in ϕ and add s as a situation argument, if one is missing

if Happy **then** laugh **else** cry $\Rightarrow$
Happy(s) $\supset Do($laugh, $s, s') \ldots$

Sometimes use '*now*' as a pseudo-situation argument to be replaced

if Happy(*now*) **then** laugh **else** cry

Knowledge in the Situation Calculus

Moore

Following Moore, introduce accessibility relation as a fluent predicate in the language: $K(s', s)$

in situation s, as far as the agent knows, s' could be the current situation

Note order of arguments, to be compatible with

$\text{Broken}(\text{obj1}, s)$, $\text{Happy}(s)$, $\text{NextTo}(x, y, s)$

↑ final argument is always the current situation

Knowledge: truth in all accessible situations

$$Know(\phi, s) \doteq \forall s' \, K(s', s) \supset \phi[s']$$

Like complex actions, knowledge is treated as a macro

Note: $Know(\text{Raining}(\underline{now}), S_0)$ expands to

$\forall s' \; K(s', S_0) \supset \text{Raining}(s')$ ✔

But: $Know(\text{Raining}(S_0), S_0)$ expands to

$\forall s' \; K(s', S_0) \supset \underline{\text{Raining}(S_0)}$?

So non-fluents and fluents that do not use the situation term '$\underline{now}$' always have known values!

Knowledge and Action

The fluent K is affected by ordinary actions and by special "knowledge-producing" perceptual actions

For ordinary actions, we have causal laws like

$$K(s', do(\text{jump}, s)) \equiv \exists s^* \; s' = do(\text{jump}, s^*) \wedge K(s^*, s))$$

the situations that are accessible after doing a jump are the images of those that were accessible before

$\therefore$ after jumping, it is known that a jump has occurred

Each knowledge-producing action is associated with a formula (or term) whose value will become known

For example, we might have the action rainchk which can be used to find out if it is raining:

$$K(s', do(\text{rainchk}, s)) \equiv \exists s^* \; s' = do(\text{rainchk}, s^*) \wedge K(s^*, s) \wedge \text{Raining}(s^*) \equiv \text{Raining}(s)$$

the situations accessible after checking are the images of those from before where Raining had the correct truth value

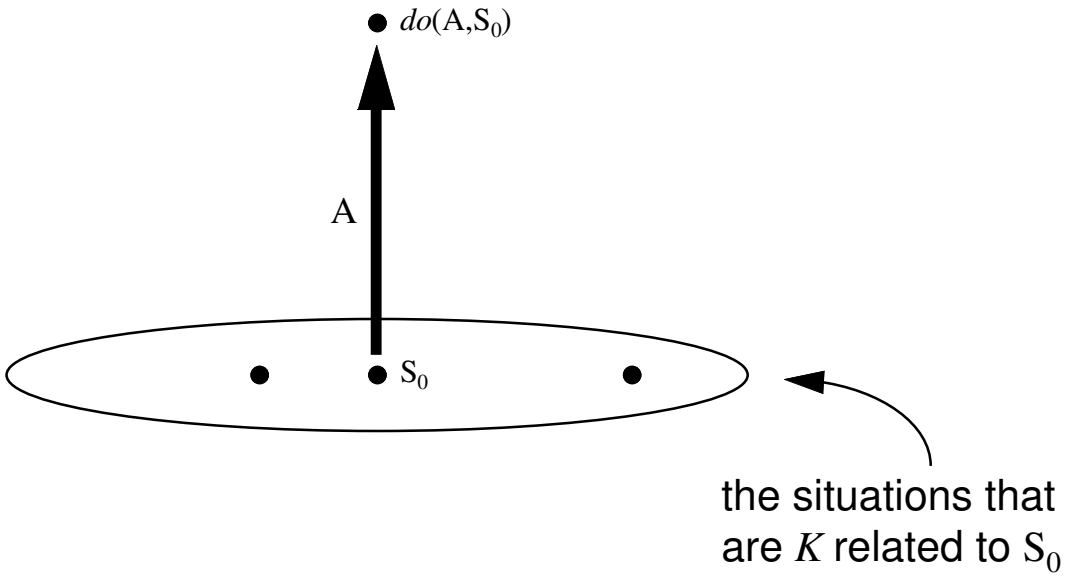
$\therefore$ after checking, it is known *whether or not* it is raining

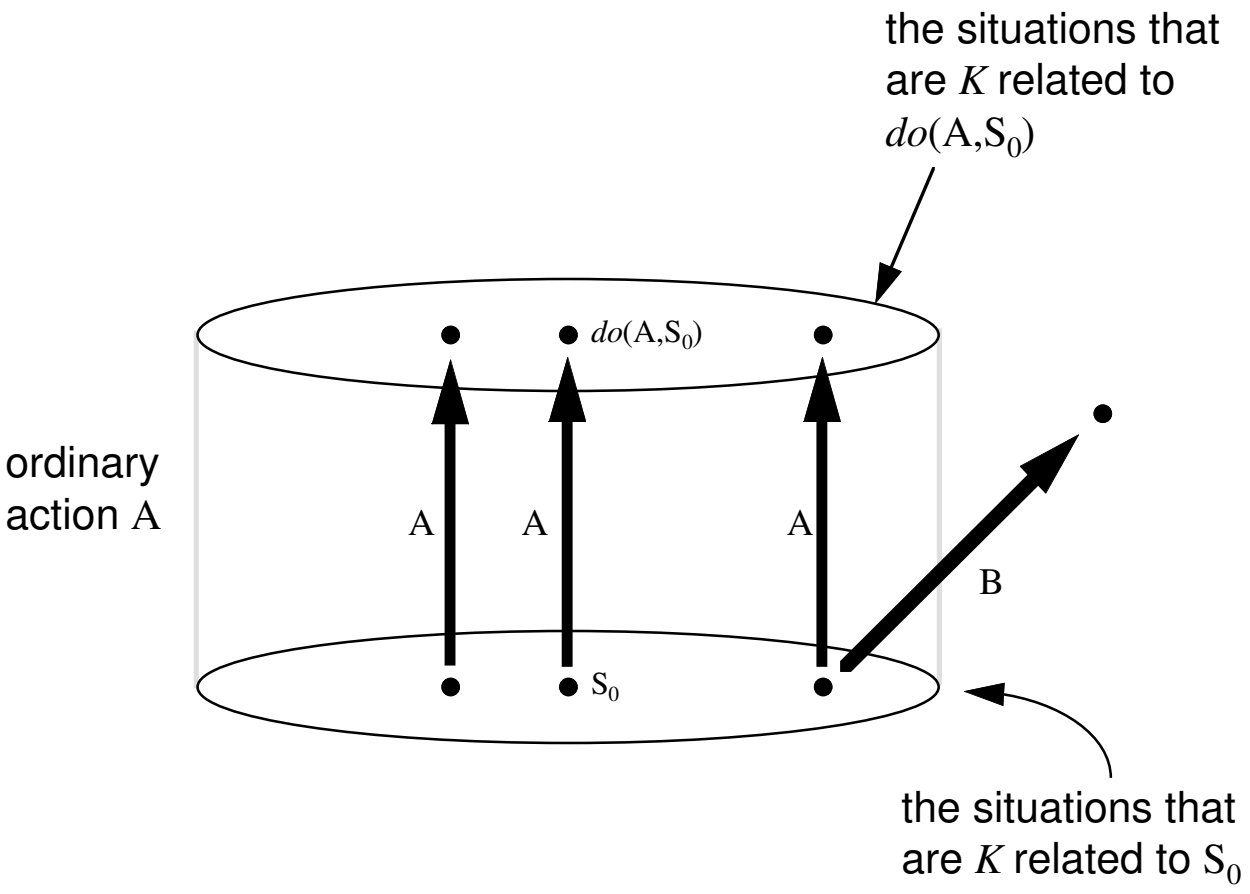
Assume n knowledge-producing actions $kp_1, ..., kp_n$. We get a successor-state axiom like

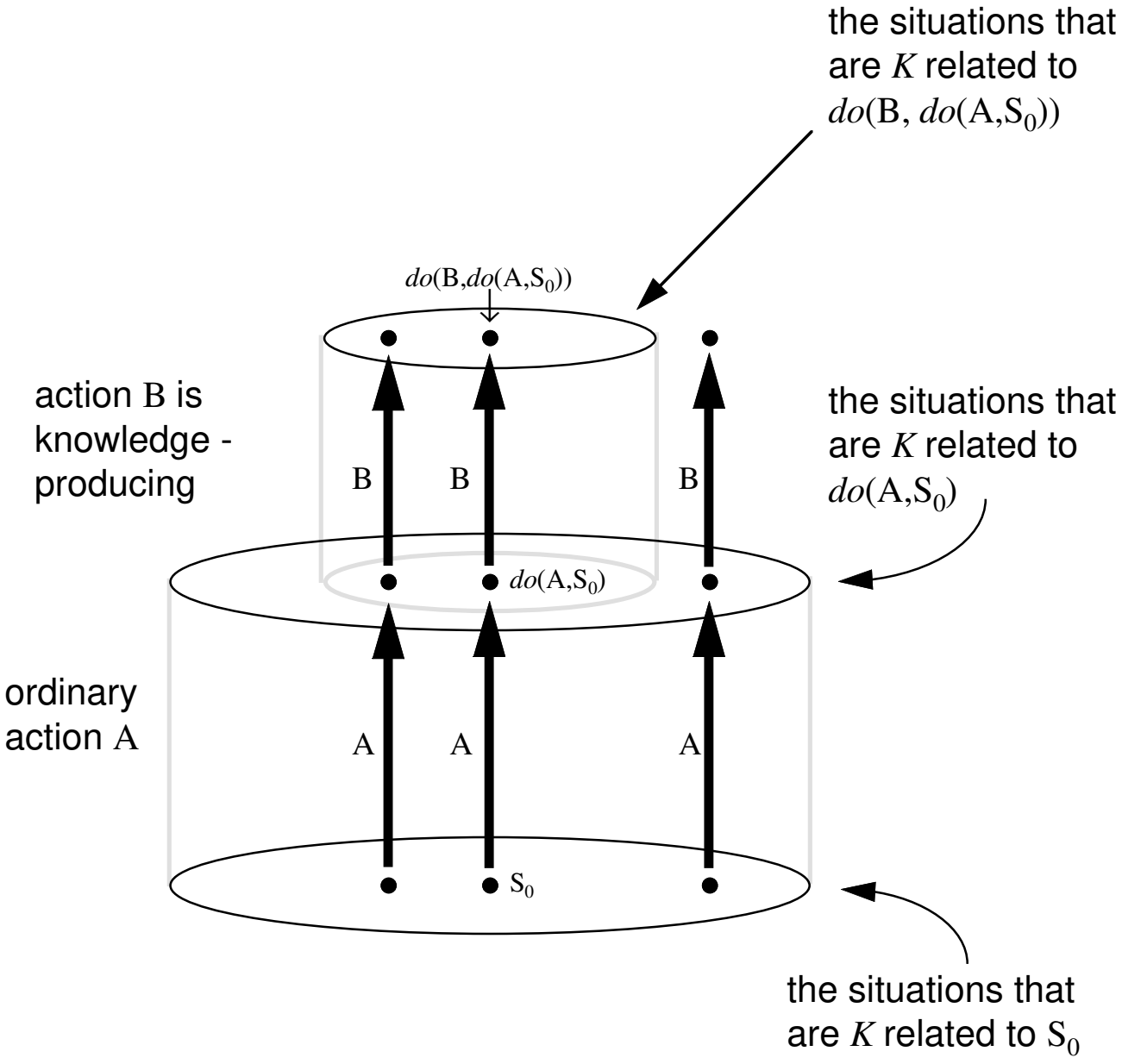
$$Poss(a,s) \supset [\, K(s', do(a,s)) \equiv \exists s^* \; s' = do(a, s^*) \wedge K(s^*, s) \wedge (a = kp_1 \supset \phi_1[s^*] \equiv \phi_1[s]) \wedge \ldots \wedge (a = kp_n \supset \phi_n[s^*] \equiv \phi_n[s])\,]$$

This completely determines K at $do(a,s)$ in terms of K at s

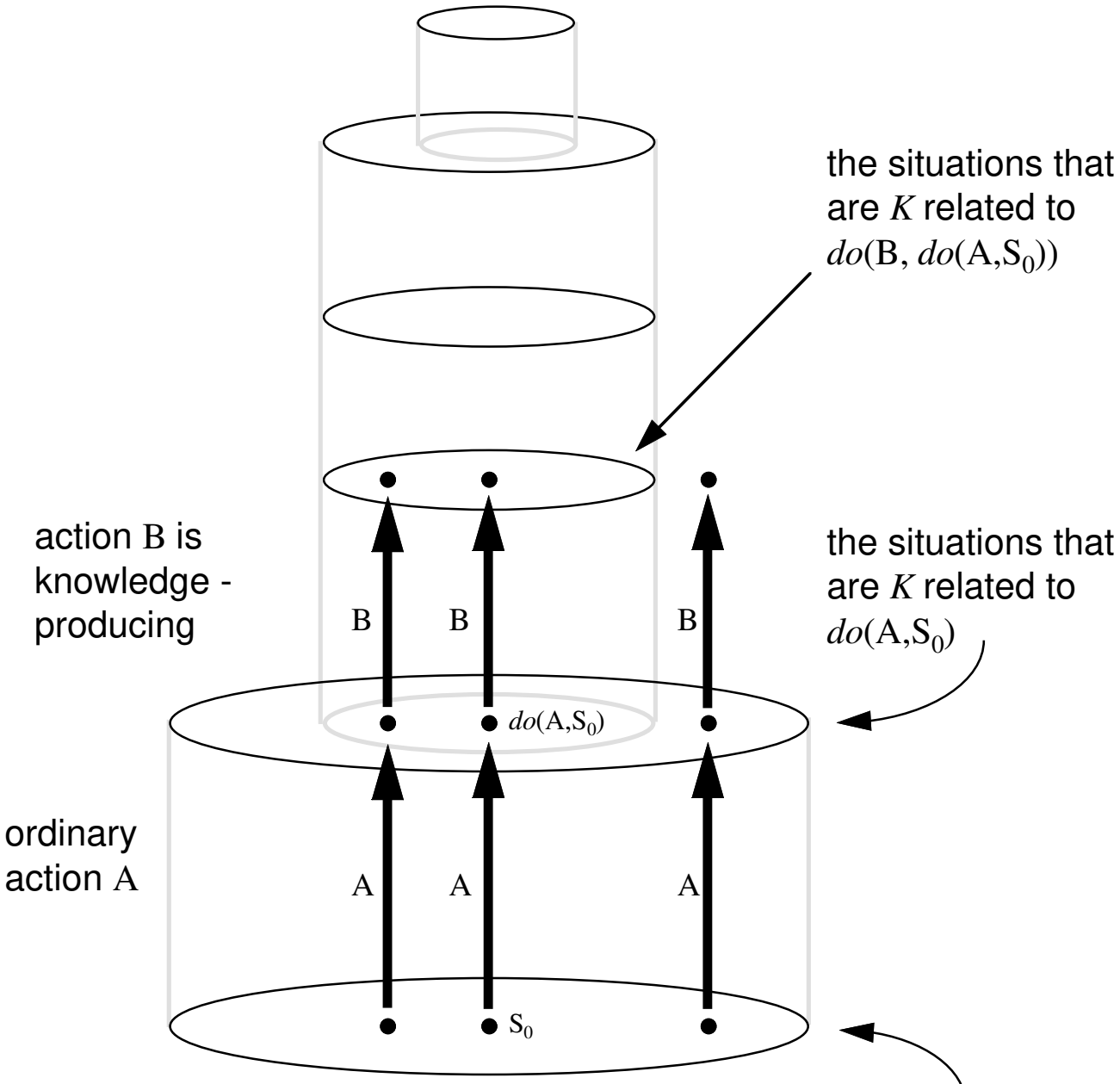
Pictorially ...




the situations that are K related to $do(A,S_0)$
$do(A,S_0)$
ordinary action A
A
A
A
B
S_0
the situations that are K related to S_0


the situations that are K related to $do(B, do(A,S_0))$
$do(B,do(A,S_0))$
action B is knowledge - producing
B
B
B
the situations that are K related to $do(A,S_0)$
$do(A,S_0)$
ordinary action A
A
A
A
S_0
the situations that are K related to S_0

Knowledge, The Picture



The K fluent at $do(a,s)$ is fully determined by a and its value at s

Plans in the Situation Calculus

Green

To achieve goal $\phi[s]$, pose $\Sigma \stackrel{?}{\models} \exists s\ \phi[s]$

then use answer extraction

Get: $s = do(a_n, do(a_{n-1}, \ldots do(a_1, S_0) \ldots))$

i.e. doing $a_1, a_2, \ldots a_n$ is sufficient to achieve ϕ

Two strong assumptions:

1. Background knowledge Σ is such that there will be a fixed sequence of actions from S_0 to a goal state

   No conditionals, loops as in

   ```
   begin
      go_outside;
      look_in_driveway;
      if Car_In_Driveway
         then drive
         else walk
   end
   ```

   Never a need for perceptual actions

2. The agent always knows enough to be able to execute each action in the sequence

   Example: want block moved to John's favourite location on the table (unchanging), given that the block can be moved anywhere.

   ϕ = At(block, favLoc(john, s), s)

   Get: a = move(block, favLoc(john, S_0))

I. Using programs

Characterize agents ability to achieve ϕ in terms of complex actions / programs

addresses problem 1 directly

$$Can(\phi, A, s) \doteq KnowHow(A, s) \wedge \forall s' Do(A, s, s') \supset \phi[s']$$

agent can achieve ϕ by doing program A iff
the agent knows how to do A, and
doing it results in a state where ϕ holds

variants: require agent to know that ϕ holds or
require agent to know 2nd conjunct *etc.*

KnowHow(A, s) is characterized by a set of axioms:

$KnowHow(t, s) \equiv \exists a\, Know(t=a, s)$, t is a prim. action term

addresses problem 2

$$KnowHow([\textbf{if } \phi \textbf{ then } A \textbf{ else } B], s) \equiv Know(\phi, s) \wedge KnowHow(A, s) \vee Know(\neg\phi, s) \wedge KnowHow(B, s)$$

$$KnowHow([A; B], s) \equiv KnowHow(A, s) \wedge Know(\forall s' Do(A, \underline{now}, s') \supset KnowHow(B, s'), s)$$

or some such

$$KnowHow([\textbf{while } \phi \textbf{ do } A], s) \equiv Know(\neg\phi, s) \vee Know(\phi, s) \wedge KnowHow(A; [\textbf{while } \phi \textbf{ do } A], s)$$

Some problems

1. Takes program text too literally

   "dumb" interpreter

   $KnowHow([\textbf{if } \phi \textbf{ then } A \textbf{ else } A], s) \supset$
   $Know(\phi, s) \vee Know(\neg\phi, s)$

2. Too weak on while loops

   $KnowHow([\textbf{while } \text{TRUE } \textbf{do } A], s)$ is not ruled out

   But then, get $Can(\text{Anything}, [\textbf{while } \text{TRUE } \textbf{do } A], s)$
   since $\forall s' \neg Do([\textbf{while } \text{TRUE } \textbf{do } A], s, s')$

   Problem: loops need a <u>least</u> fixed point

3. How to get $Can(\phi, s)$?

   One way: $\exists\, Program{:}x \;\; Can(\phi, x, s)$

   But this requires *quantifying over programs*

   Conflicts with our solution to frame problem

However, this approach gets at the main issue:

the agent must always know
what primitive action to do next

Want: at each step s, $\exists a\, Know($'should do a now', $s)$

The tree example

Assume that a certain tree will fall down after some number of primitive chopping actions

$$\text{Down}(s) \doteq \text{magicNum}(s) = 0$$

$$\text{magicNum}(do(a,s)) = k \equiv$$
$$a = \text{chop} \wedge \text{magicNum}(s) = k+1 \vee$$
$$a \neq \text{chop} \wedge \text{magicNum}(s) = k$$

succ state axiom

and that the agent can always find out by looking whether or not the tree is down.

$$K(s', do(\text{look}, s)) \equiv \exists s^* \; s' = do(\text{look}, s^*) \wedge K(s^*, s) \wedge \text{Down}(s) \equiv \text{Down}(s^*)$$

$$\therefore \quad \forall s \; Know(\text{Down}, do(\text{look}, s)) \vee Know(\neg\text{Down}, do(\text{look}, s))$$

But we do not assume that the agent knows

- how many chop actions are necessary: $magicNum(S_0)$
- which chop will be the final action

Want a definition of *Can* such that $Can(\text{Down}, S_0)$.

Why? Intuitively, by doing program choploop:

```
look;
while ¬Down do
      chop;
      look
end while
```

II. Iterative version

$Can(\phi, s) \doteq \exists k\ Can^k(\phi, s)$ where

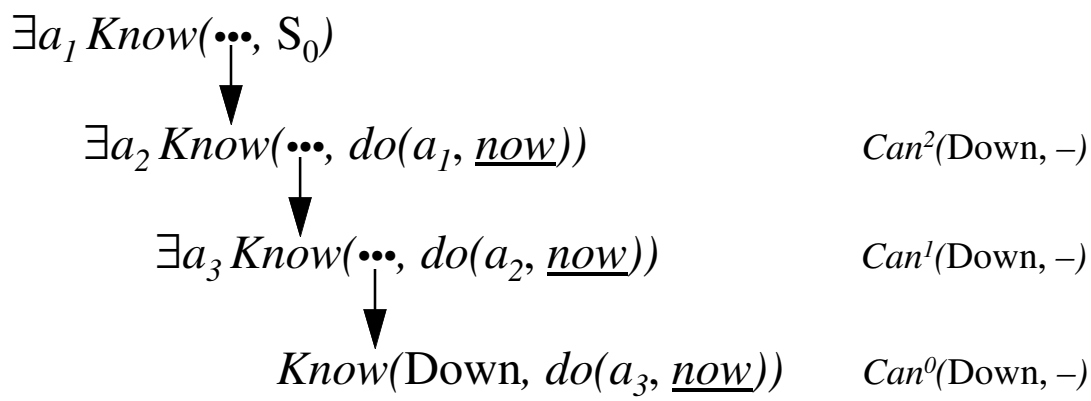
$Can^0(\phi, s) \equiv Know(\phi, s)$

$Can^{k+1}(\phi, s) \equiv \exists a\ Know(Can^k(\phi, do(a, \underline{now})), s)$

Handles many (most? all?) examples that do not use unbounded while loops

However, it fails on the chopping-tree example

for example, $Can^3(\text{Down}, S_0)$ is equivalent to



So Can^k requires knowing initially that k steps are sufficient to achieve the goal

Instead of $\underline{\exists k}$ ... $\exists a\ Know(Can^k(\bullet\bullet\bullet), s)$,

would prefer ... $\exists a\ Know(\underline{\exists k}\ Can^k(\bullet\bullet\bullet), s)$,

which is just ... $\exists a\ Know(Can(\bullet\bullet\bullet), s)$

III. Fixed-point version

Would like a definition of Can that satisfies

1. $Know(\phi, s) \supset Can(\phi, s)$
2. $\exists a\ Know(Can(\phi, do(a,\underline{now})), s) \supset Can(\phi, s)$

Try characterizing Can by axiom:

$$Can(\phi, s) \equiv Know(\phi, s) \vee \exists a\ Know(Can(\phi, do(a,\underline{now})), s)$$

done

know what to do next

Problem: too weak

$Can(\phi, s) \equiv \text{TRUE}$ is consistent with this axiom

Better idea: use *least* fixed point

Idea: $Can_{\perp}(\phi, s)$ holds iff $C(s)$ where C is the *smallest* predicate that satisfies the axiom

Formally: $Can_{\perp}(\phi, s) \doteq \forall C\ (\bullet\bullet\bullet \supset C(s))$

$$\forall s'\ C(s') \equiv [Know(\phi, s') \vee \exists a\ Know(C(do(a,\underline{now})), s')]$$

Problem: too strong!

Fails on tree example: $\neg Can_{\perp}(\text{Down}, S_0)$

Why? consider $D(s)$ defined as $Know(\text{Down}, s)$

Then D is a fixed point, given $\neg Know(D(do(a,\underline{now})), s')$

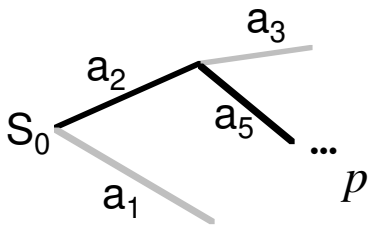
But $Can_{\perp}$ must hold in no more situations than D

Thus: $\neg Can_{\perp}(\text{Down}, S_0)$, unless $Know(\text{Down}, S_0)$

Final version: path-based

Path: provides a mapping from situations to actions

– intuitively: the next action to perform



$next(p, S_0) = a_2$

$next(p, do(a_2, S_0)) = a_5$

...

Characterized by a 2nd-order axiom:

$$\forall f{:}[sit \rightarrow action]\ \exists p{:}Path\quad \forall s\ next(p, s) = f(s)$$

Might also require: $Poss(next(p, s), s)$ the actions on p are physically possible

$OnPath(p, s, s')$ means path p goes from s to s' at least

$$OnPath(p, s, s') \doteq s \leq s' \wedge \forall a \forall s^* (s < do(a, s^*) \leq s') \supset next(p, s^*) = a$$

Useful for

- time, occurrences
- correctness of non-terminating programs

  fairness, etc.

Definition of CAN

$CanFollow(p, s, s') \doteq$

$OnPath(p, s, s') \wedge$

$\forall s^* (s \leq s^* \prec s') \supset \exists a\ Know(a=next(p, \underline{now}), s^*)$

at every step between s and s', the agent knows the value of p at that situation

$\therefore$ the agent knows how to stay on the path

Notation: goals ϕ to be achieved may use <u>two</u> pseudo-situation arguments

- the initial state, written '*<u>start</u>*' (often S_0)
- the desired goal state, written '*<u>end</u>*'

Example: $\forall x\ \mathrm{On}(x,\mathrm{table},\underline{start}) \supset \mathrm{In}(x,\mathrm{box},\underline{end})$

get everything on the table into the box

$$CanGet(\phi, p, s) \doteq \exists s'\ CanFollow(p, s, s') \wedge \phi[s, s']$$

starting at s, the agent can follow p to a state s' where ϕ holds

$$Can(\phi, s) \doteq \exists p\ Know(CanGet(\phi, p, \underline{now}), s)$$

Two variants: $\exists p\ CanGet(\phi, p, s)$ agent unaware

$Know(\exists p\ CanGet(\phi, p, \underline{now}), s)$??

Some general properties*

1. *Can* is implied by the iterative version

   $\exists k\ Can^k(\phi, s) \supset Can(\phi, s)$

   – If know how many steps are required and what to do at each step, then $Can(\phi, s)$.

   – But not conversely, (cf. tree example below)

2. *Can* satisfies the two properties of the fixed-point

   1. $Know(\phi, s) \supset Can(\phi, s)$
   2. $\exists a\ Know(Can(\phi, do(a,\underline{now})), s) \supset Can(\phi, s)$

      – Thus, it is implied by the least-fixed point

      $$Can_{\perp}(\phi, s) \supset Can(\phi, s)$$

      – But again not conversely, (cf. tree example)

*Hedge:

In the following it is assumed that the K relation is transitive and Euclidian. However, it is not required to be reflexive.

Thus, **weak S5** (*aka* **K45**), which avoids the difficulties of **strong S5** regarding introspection, may be used.

Also, for simplicity, ϕ is assumed not to use the *start* parameter

CAN and Programs

Moore's version of CAN depended on programs and an auxilliary definition of *KnowHow(A,s)*

Here: we can do the converse!

$$KnowHow(A, s) \doteq Can(Do(A, \underline{start}, \underline{end}), s)$$

note use of *start*

Some theorems:

Assume that $\forall s\, KnowHow(A, s) \wedge KnowHow(B, s)$

- $KnowIf(\phi, s) \supset KnowHow([\mathbf{if}\ \phi\ \mathbf{then}\ A\ \mathbf{else}\ B], s)$

  like Moore's account

- $KnowHow([\mathbf{if}\ \phi\ \mathbf{then}\ A\ \mathbf{else}\ A], s)$

  smart interpreter, unlike Moore's

- $Know(\exists s'\, Do([\mathbf{while}\ \phi\ \mathbf{do}\ A], \underline{now}, s'), s) \supset KnowHow([\mathbf{while}\ \phi\ \mathbf{do}\ A], s)$

  smart interpreter can just do *A* forever

  (ability to "stop" not required)

But <u>not</u>

- $Know(\exists s'\, Do([\mathbf{while}\ \phi\ \mathbf{do}\ A], \underline{now}, s'), s) \supset KnowHow([\mathbf{while}\ \phi\ \mathbf{do}\ A]; B, s)$

  need to know when done loop to do *B*

The tree example revisited

Theorem: $Can(Know(\text{Down}, \underline{end}), S_0)$ can fell the tree (& stop)

Proof:

Let path p be defined by $next(p, s) = \text{chop}$, if $s = do(\text{look}, -)$
$next(p, s) = \text{look}$, otherwise

Claim: $\forall s \; \exists a \; Know(a = next(p, \underline{now}), s)$

Observe: by succ. state axiom for K, we have that
$s = do(\text{look}, -)$ iff $\forall s' \; K(s', s) \supset s' = do(\text{look}, -)$

So, if $s = do(\text{look}, -)$, then $\forall s' \; K(s', s) \supset next(p, s') = \text{chop}$,
and otherwise, $\forall s' \; K(s', s) \supset next(p, s') = \text{look}$

Let s_1 be any situation such that $K(s_1, S_0)$.

Claim: $CanGet(Know(\text{Down}, \underline{end}), p, s_1)$

We have that $\exists s' \; Do(\text{choploop}, s_1, s')$,
since $\text{magicNum}(s_1)$ chops are always sufficient

So $\exists s' \; OnPath(p, s_1, s') \wedge \text{Down}(s') \wedge s' = do(\text{look}, -)$

and thus: $\exists s' \; CanFollow(p, s_1, s') \wedge Know(\text{Down}, s')$

Since s_1 was chosen arbitraily, we have that
$Know(CanGet(Know(\text{Down}, \underline{end}), p, \underline{now}), S_0)$ ■

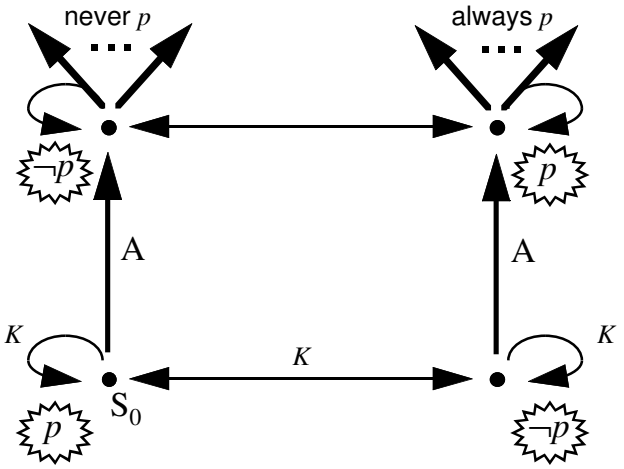
Note: proof requires knowing whether tree is down.
But without this, we still get $Can(\text{Down}(\underline{end}), S_0)$

agent knows that chopping forever will achieve it!

Fixed-points reconsidered

Can does <u>not</u> satisfy the fixed-point equivalence

$$Can(\phi, s) \equiv Know(\phi, s) \vee \exists a\, Know(Can(\phi, do(a, \underline{now})), s)$$



counter-model

$Can(p, S_0)$ but
$\neg Know(p, S_0)$
$\neg Know(Can(p, do(A, \underline{now})), S_0)$

However, it does* when the goal is of the form $Know(\psi)$

Let $Can^+(\phi, s) \doteq Can(Know(\phi, \underline{end}), s)$

Then $Can^+(\phi, s) \equiv Know(\phi, s) \vee \exists a\, Know(Can^+(\phi, do(a, \underline{now})), s)$

This can also be generalized* to any goal ϕ for which there is a sense action A:

$$\forall s\ Know(\phi, do(A, s)) \vee Know(\neg\phi, do(A, s))$$

*Restrictions on K apply

Are programs necessary?

Having a program is <u>sufficient</u> for Can:

$$KnowHow(A, s) \wedge Know(\bullet\bullet\bullet, s) \supset Can(\phi, s)$$

$$\forall s' Do(A, \underline{now}, s') \supset \phi[\underline{now}, s']$$

doing A results in a situation where ϕ holds

Follows from:

$$Can(\psi, s) \wedge Know(\bullet\bullet\bullet, s) \supset Can(\phi, s)$$

$$\forall s' \; \underline{now} \leq s' \wedge \psi[\underline{now}, s'] \supset \phi[\underline{now}, s']$$

ψ implies ϕ in all future states

Does the converse hold?

Is it the case that if $Can(\phi, s)$, there must be some implicit program A that determines the path to follow?

If not, then perhaps the language is too strong.

Planning revisited: $\Sigma \models Can(\phi, s)$

constructive proof for $\Sigma \models \exists p \; Know(\bullet\bullet\bullet, s)$?

from which can we <u>derive</u> a program ??