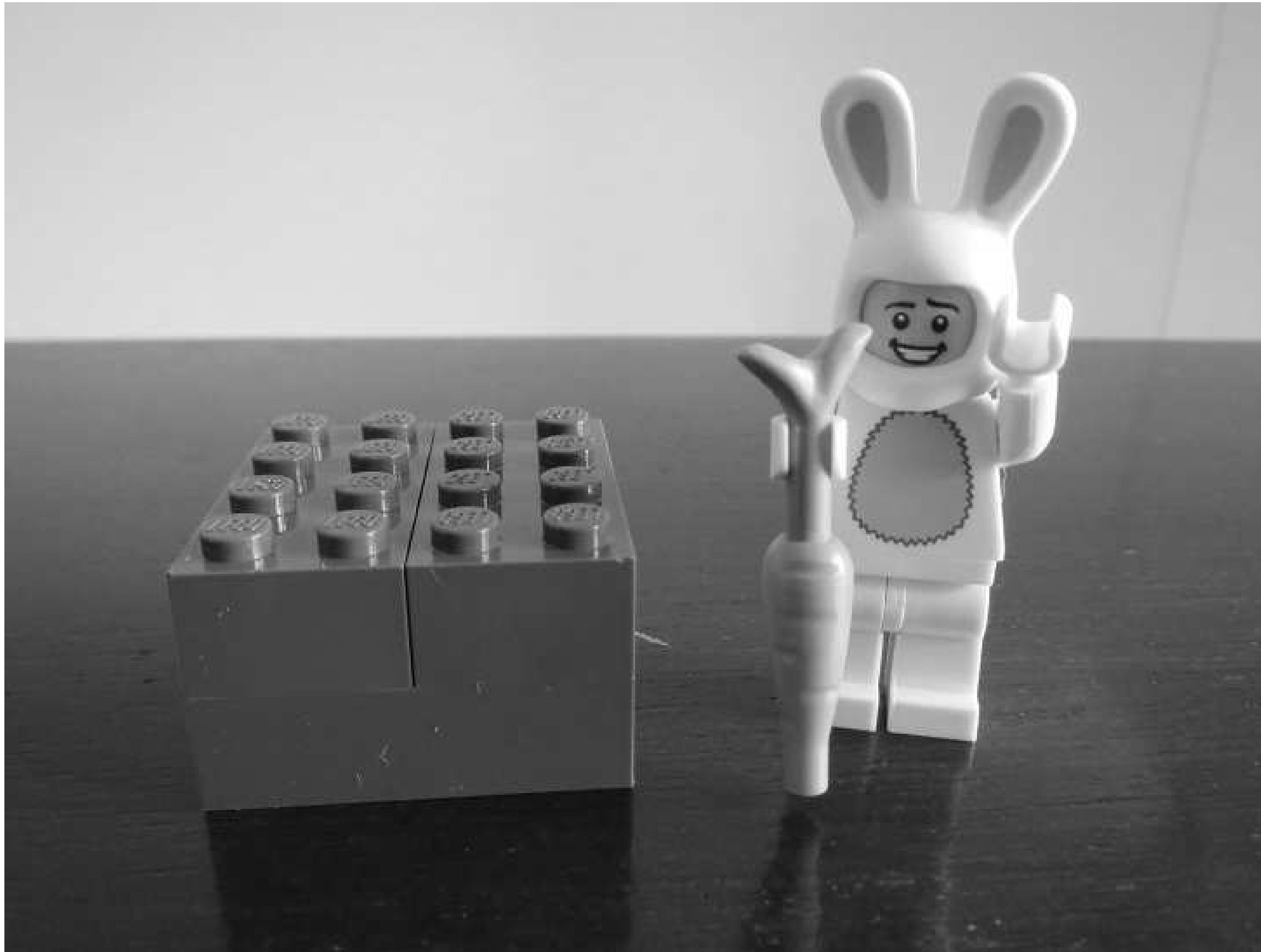
Actions and Change



Outline

- So far we've dealt with static KB
- Why have a dynamic KB?
- Action and change
- Frame problems
- Simple state-changes
- STRIPS: add, delete and CWA
- Action logic: situation calculus
- The gamut of action logics

Relations and Time

- Agents reason in time and about time
- Time often implicit (ordered snapshots)
- When modeling relations, two types exist:
- Static relations are those relations whose values does not depend on time
- Dynamic relations are relations whose truth values depends on time:
  - derived relations whose definition can be derived from other relations for each time
  - primitive relations whose truth value can be determined by considering previous times

(Related to the distinction between Bayesian networks and dynamic Bayesian networks)

Dynamics: what and how

Two types of changing beliefs:

- In case the beliefs themselves are dynamic, then the general area is called belief revision which generally deals with: let φ be new information, and B a belief state, what is $B \cup \varphi$? (we will not pursue this direction in this course)
- In case the world itself can change, and we want to model how it changes (and thus how we should change our beliefs) then we need an action formalism (topic of this lecture)

Generally fluents are predicates (or functions) whose values may vary from situation to situation. Many ways to model this in logic:

- assert (believe) and retract (forget) logical facts (Prolog: `assert` and `retract`)
- use a datastructure (term) to keep the *current* state or model
- enrich each logical fact with a time stamp (e.g. $\mathtt{lightOn}(\mathtt{lamp}_1, \mathtt{time}_n)$)
- consider multiple models (seen in e.g. nonmonotonic logic, fixed points)
- fully axiomatize and use *situations* (just a sequence of actions performed)
- ...

Cold Turkey?

Let there be a turkey and a gun. The turkey is named Fred and with a loaded gun Fred can be killed. At the start of the situation, Fred is alive and the gun is not loaded. Consider three possible actions one can take:

- load: load the gun
- wait: just some action without effects (e.g. for steady aim)
- shoot: use the loaded gun to shoot at Fred

Original formulation by Hanks and McDermott (1987) to show problems with logical formulations of reasoning about action and change.
McCarthy (1986) proposed circumscription (for solving the frame problem):

$$Holds(p,s) \wedge \neg abormal(p,a,s) \rightarrow Holds(p,do(a,s))$$

Logical Turkey = Dead Turkey

A policy for diner is: *load* $\rightarrow$ *wait* $\rightarrow$ *shoot*

The simplest logical formulation of this problem is (only formalizing the changes):

$$\begin{array}{l} alive(0) \\ \neg loaded(0) \\ true \rightarrow loaded(1) \quad (action: load) \\ loaded(2) \rightarrow \neg alive(3) \quad (action: shoot) \end{array}$$

Minimization of the changes gives a plausible model:

$$\begin{array}{l} alive(0)\;\; alive(1)\;\; alive(2)\;\; \neg alive(3) \\ \neg loaded(0)\;\; loaded(1)\;\; loaded(2)\;\; loaded(3) \end{array}$$

A consistent model with only two changes.

Warm Turkey?

Minimization of the changes gives *another* model:

> *alive(0) alive(1) alive(2)* $\neg$*alive(3)*
> $\neg$*loaded(0) loaded(1)* $\neg$*loaded(2)* $\neg$*loaded(3)*

This model too has only two changes. Fred's alive now!?

Apparently, the wait action has mysteriously unloaded the gun.

Compare to diagnostic reasoning (**minimal** diagnoses, hitting sets, conflict sets) and nonmonotic reasoning.

The General Frame Problem

The YSP (*Yale Shooting Problem*) became a big motivation for much research on the so-called **Frame Problem**

- *How to specify what does **not** change in a logical system when actions are applied*

Modern solutions: separate the specification of the effects of actions from the task of reasoning about these actions.

We will review three classes of solutions in the remainder of this lecture:

- Explicit state change operators
- STRIPS planning and the closed world assumption
- Action logics: situation calculus

More big problems

The frame problem (FP) is about things that do not change.

- **representational FP**: how to represent the frame axioms?
- **inferential FP**: how to compute the outcome of sequences of actions?

Two related problems:

- **ramification problem**: if an action changes a fluent *indirectly*, how to represent that and reason about it? For example, $\mathit{Bring}(\mathit{briefcase}, \mathit{here})$ changes the briefcase's locaton to *here*, but if $\mathit{In}(\mathit{pen}, \mathit{briefcase})$ then the truth value of $\mathrm{At}(\mathit{pen}, \mathit{here})$ changes too.
- **qualification problem**: how to specify *all* things relevant for the applicability of an action? For example, we can shoot the gun, *unless* it's jammed, *unless* it is a water gun, *unless* it transforms into a magical rabbit if touched, *unless* ... etc.

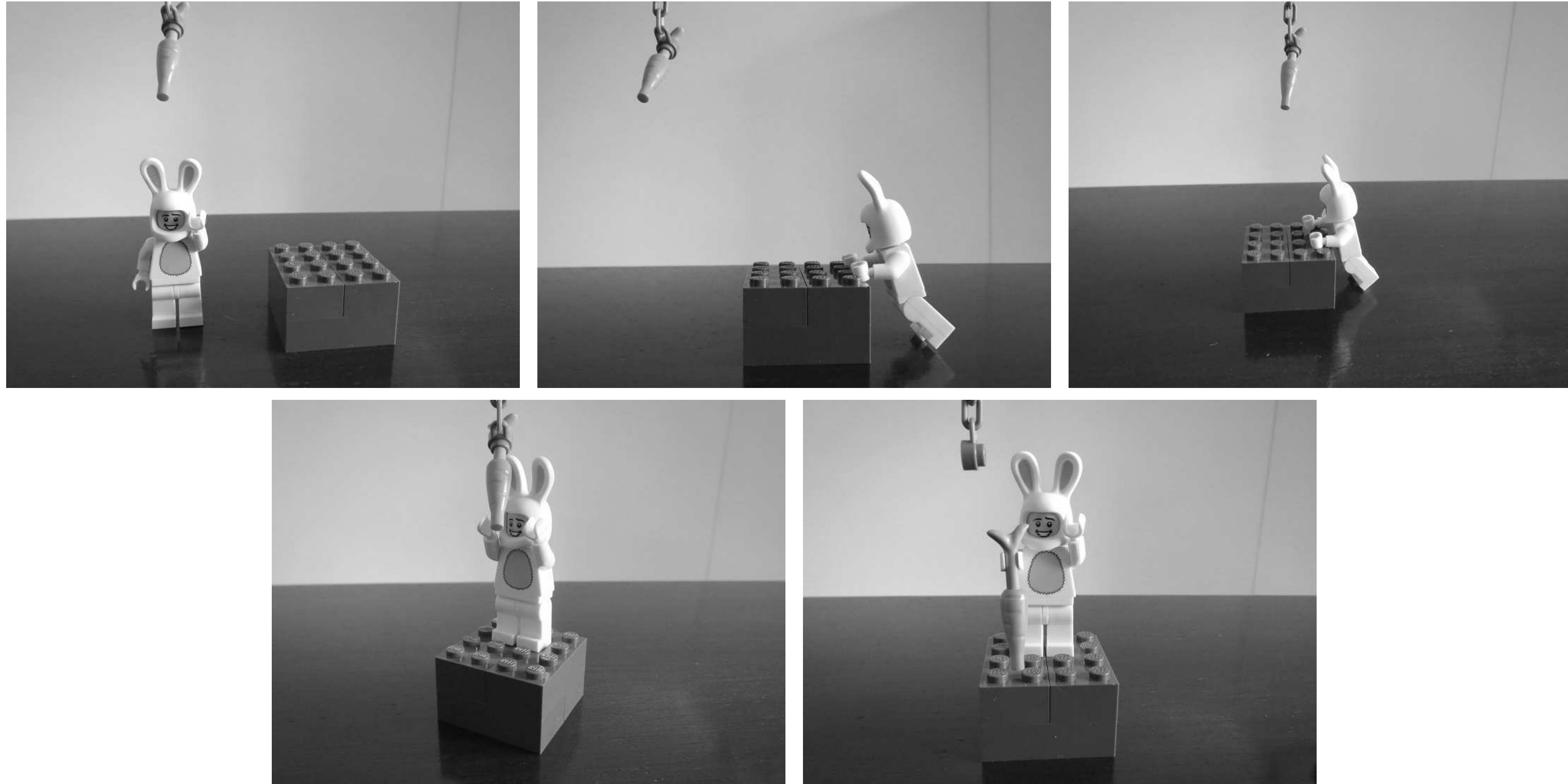
In the remainder we will mainly deal with the frame problem.

(Model 1) Monkey-Banana

A classic (McCarthy): A hungry monkey: *A monkey is in a room where a bunch of bananas is hanging from the ceiling, too high to reach. In the corner of the room is a box, which is not under the bananas. The box is sturdy enough to support the monkey if he climbs on it, and light enough so that he can move it easily. If the box is under the bananas, and the monkey is on the box, he will be high enough to reach the bananas.* Initially the monkey is on the ground, and the box is not under the bananas. There's a lot the monkey can do:

- Go somewhere else in the room (assuming the monkey is not standing on the box)
- Climb onto the box (assuming the monkey is at the box, but not on it)
- Climb off the box (assuming it is standing on the box)
- Push the box anywhere (assume monkey is at box, but not on it)
- Grab the bananas (assume monkey is on the box, under bananas)

Actions and Change



Logic is syntax and semantics...
works equally fine with rabbits and carrots...

Monkey-Banana

To automaticallly find a plan we need to formalize it as a search problem with four elements:

- **States**: a form of *snapshots* of how the world can look like. A state consist of the locations of the monkey, the box and the bananas.
- **Operators**: actions that can change the state of the world. Moving the box changes the location of the box (and the monkey).
- **Initial State**: the state of the world at the start of problem. The monkey is not on the box, and the box is not under the bananas.
- **Goal State**: the *desired* state we want to be in. This state is the goal of the whole planning process.

Planning consists of computing a *sequence of operators* such that once that sequence is applied starting from the initial state, one will reach the goal state.

Planning and Search

A simple logical model: each state has the location of the bananas (b), the monkey (m) and the box (l), as well as whether the monkey is on the box (o, is $\mathtt{y}$ or $\mathtt{n}$), and whether the monkey has the bananas h ($\mathtt{y}$ or $\mathtt{n}$).

Idea: use Prolog *lists of atoms*, e.g. $[\mathtt{loc1}, \mathtt{loc2}, \mathtt{loc3}, \mathtt{n}, \mathtt{n}]$
A goal state will (at least) have as last element in the list a '$\mathtt{y}$'.

```
initial_state([loc1, loc2, loc3, n, n]).
goal_state([_, _, _, _, y]).
legal_move([B, M, M, n, H], climb_on, [B, M, M, y, H]).
legal_move([B, M, M, y, H], climb_off, [B, M, M, n, H]).
legal_move([B, B, B, y, n], grab, [B, B, B, y, y]).
legal_move([B, M, M, n, H], push(X), [B, X, X, n, H]).
legal_move([B, _, L, n, H], go(X), [B, X, L, n, H]).
```

Note that the representation here is fixed in size and order.

State State Planning

A **general planning algorithm in Prolog**

$\mathtt{plan(L) :- initial_state(I), goal_state(G), reachable(I, L, G).}$

$\mathtt{reachable(S, [], S).}$

$\mathtt{reachable(S1, [M|L], S3) :- legal_move(S1, M, S2), reachable(S2, L, S3).}$

Running it naively is not the best idea:

$? - \mathtt{plan(P).}$

$\mathtt{ERROR : Outoflocalstack}$

Better take a fixed length:

$? - \mathtt{plan([X, Y, Z, W]).}$

$\mathtt{X = go(loc3),}$

$\mathtt{Y = push(loc1),}$

$\mathtt{Z = climb_on,}$

$\mathtt{W = grab;}$

$\mathtt{false.}$

State State Planning

Easy extension: iterative deepening search (many others possible)

```
bplan(L) :- tryplan([],L).
tryplan(L,L) :- plan(L).
tryplan(X,L) :- tryplan([_|X],L).
```

```
1 ?- bplan(L).
L = [go(loc3), push(loc1), climb_on, grab] ;
L = [go(loc3), push(loc1), climb_on, grab, climb_off] ;
L = [go(loc3), push(_G228), push(loc1), climb_on, grab] ;
L = [go(loc3), push(loc1), go(loc1), climb_on, grab] ;
L = [go(_G211), go(loc3), push(loc1), climb_on, grab] ;
L = [go(loc3), climb_on, climb_off, push(loc1), climb_on, grab] ;
L = [go(loc3), push(loc1), climb_on, climb_off, climb_on, grab] ;
L = [go(loc3), push(loc1), climb_on, grab, climb_off, climb_on] ;
...
L = [go(loc3), push(_G231), push(loc1), climb_on, grab, climb_off] ;
L = [go(loc3), push(_G231), push(_G248), push(loc1), climb_on, grab] ;
L = [go(loc3), push(_G231), push(loc1), go(loc1), climb_on, grab] ;
...
L = [go(loc3), push(loc1), go(_G248), go(loc1), climb_on, grab] ;
L = [go(_G214), go(loc3), push(loc1), climb_on, grab, climb_off] ;
...
```

(Model 2) The STRIPS representation

- STRIPS is a representation language for planning problems
- Originally developed for a robot named Shakey in the sixties
- Whereas our previous state-operator based model represents explicit transitions between states, STRIPS defines operators that syntactically transform world models.
- A single world state exists at each time, represented by a database of ground atomic wffs (e.g. *in(robot,room)*)
- we cannot reason directly about actions (it is not a logic) since the actions are not part of the logical world model (e.g. they are defined procedurally).
- STRIPS does not keep track of the history; at each moment in time there is only one state.

STRIPS operators

A STRIPS operator $\langle Act, Pre, Add, Del \rangle$ features four components:

- action name *Act*: the name (plus arguments) of the action described in the operator
- precondition *Pre*: atoms that must be true in order to apply the action
- delete list: *Add*: atoms to be deleted from the current state (those becoming `false`) if the action is applied
- add list: *Del*: atoms to be added to the current state (those becoming `true`) if the action is applied

STRIPS operators (2)

Let O be an operator and let S be a state, i.e. a set of ground relational atoms. The *operational semantics* of applying O to S is

- first find a *matching* of Pre and S, i.e. find a subset $S' \subseteq S$ and a substitution θ such that *Pre*$\theta \equiv S'$
- compute the new state as $S'' = (S \backslash \mathit{Del}\theta) \cup \mathit{Add}\theta$.

Example: Let $O = \langle \mathit{Go}(x, y)$
$\{\mathit{At}(\mathit{Monkey}, x), \mathit{On}(\mathit{Monkey}, \mathit{Floor})\}$,
$\{\mathit{At}(\mathit{Monkey}, x)\}, \{\mathit{At}(\mathit{Monkey}, Y)\}\rangle$,
and $S = \{\mathit{On}(\mathit{Monkey}, \mathit{Floor}), \mathit{At}(\mathit{Monkey}, \mathit{Loc1}), \ldots, etc.\}$

Taking $\mathit{Go}(\mathit{Loc1}, \mathit{Loc2})$ spawns the new state
$S' = \{\mathit{On}(\mathit{Monkey}, \mathit{Floor}), \mathit{At}(\mathit{Monkey}, \mathit{Loc2}), \ldots, etc.\}$

STRIPS operator example

In Prolog this looks like this:

```
action(go(X,Y), [at(monkey,X), on(monkey,floor)],
    [at(monkey,X)], [at(monkey,Y)]).

action(push(B,X,Y),
    [at(monkey,X), at(B,X), on(monkey,floor), on(B,floor)],
    [at(monkey,X), at(B,X)], [at(monkey,Y), at(B,Y)]).

action(climbon(B),
    [at(monkey,X), at(B,X), on(monkey,floor), on(B,floor)],
    [on(monkey,floor)], [on(monkey,B)]).

action(grab(B),
    [on(monkey,box), at(box,X), at(B,X), status(B,hanging)],
    [status(B,hanging)], [status(B,grabbed)]).
```

STRIPS progressive planning

```
plan(State, Goal, Plan):-
    plan(State, Goal, [], Plan).

plan(State, Goal, Plan, Plan):-
  is_subset(Goal, State), nl,
  write_sol(Plan).

plan(State, Goal, Sofar, Plan):-
  action(A, Preconditions, Delete, Add),
  is_subset(Preconditions, State),
  \+ member(A, Sofar),
  delete_list(Delete, State, Remainder),
  append(Add, Remainder, NewState),
  plan(NewState, Goal, [A|Sofar], Plan).

test1(Plan):-
    plan([on(monkey,floor),on(box,floor),at(monkey,loc1),at(box,loc2),
           at(bananas,loc3),status(bananas,hanging)],
    [status(bananas,grabbed)],
                 Plan).
```

Grabbing Bananas

```
3 ?- test1(P).
go(loc1,_G209)
push(box,loc2,_G249)
climbon(monkey)
climbon(box)
grab(bananas)

go(loc1,loc2)
push(box,loc2,loc3)
climbon(box)
grab(bananas)
P = [grab(bananas), climbon(box), push(box, loc2, loc3), go(loc1, loc2)] ] ;
climbon(monkey)
climbon(box)
climbon(monkey)
false.
```

Wrong initial situation

```
test2(Plan):-
    plan([on(floor,monkey),on(box,floor),at(monkey,loc1),at(box,loc2),
           at(bananas,loc1),status(bananas,hanging)],
    [status(bananas,grabbed)],
                 Plan).

4 ?- test2(P).
false.
```

Climbing

```
test3(Plan):-
    plan([on(monkey,box),on(box,floor),at(monkey,loc1),at(box,loc1),
          at(bananas,loc2),status(bananas,hanging)],
    [status(bananas,grabbed)],
                Plan).

action(climboff(B),
    [at(monkey,X), at(B,X), on(monkey,B), on(B,floor)],
    [on(monkey,B)],
    [on(monkey,floor)]).

climboff(box)
go(loc1,loc1)
push(box,loc1,loc2)
climbon(box)
grab(bananas)
P = [grab(bananas), climbon(box), push(box, loc1, loc2), go(loc1, loc1), climboff(b
```

About STRIPS

- STRIPS is practical, and prototypical for many **action planning languages** such as ADL (action description language, Pednault) and PDDL (planning domain description language)
- Many forms of planning are easy to adapt to this format (regression planning, HTN, ABSTRIPS, Graphplan, etc.)
- International planning competition (IPC) uses extensions of PDDL.
- Probabilistic aspects can be added in various ways (e.g. to form relational Markov decision processes)
- Frame problem solution: procedural meaning of the actions (i.e. how to apply them) and closed world assumption (CWA).
- For each action there is one rule (deterministic worlds), but rules quickly grow with the number of fluents, ramifications, etc. (e.g. to *move* block a onto b the operator needs to know explicitly that a was on c in order to *delete On*(a, c) and *add Clear*(c).
- does not support general reasoning about the domain and the actions (it is not a logic, search for semantics topic of much research)

(Model 3) Situation calculus

The situation calculus is a FOL system for representing changing worlds, where these changes are usually triggered by named actions.
There are two main *sorts* in the logic:

- **actions**: such as
  - *put*(x, y)
  - *walk*(loc)
  - *pickup*(r, x)
- **situations**: denoting possible world *histories*. A distinguished constant S_0 and function symbol *do* are used:
  - S_0: the *initial situation* (before any actions have been performed)
  - *do*(s, a): the situation that results from doing action a in situation s

For example *do*(*put*(A, B), *do*(*put*(B, C), S_0))
(situation resulting from putting A on B after putting B on C in the initial situation)

Fluents in the SC

Predicates or functions whose values may vary from situation to situation are called *fluents*

These are written using predicate or function symbols whose last argument is a situation

for example *Holding*(r, x, s): robot r is holding object x in situation s
can have $\neg$*Holding*$(r, x, s) \wedge$ *Holding*$(r, x,$ *do*$($*pickup*$(r, x), s)$
(the robot is not holding the object x in situation s, but is holding it in the situation that results from picking it up)

Note: there is no distinguished *current* situation. A sentence can talk about any situation, past, present or future.

A distinguished predicate symbol *Poss*(a, s) is used to state that s may be performed in situation s, e.g.

Poss$($*pickup*$(r, x), S_0)$
(robot r can indeed pick up object x in the initial situation)

Preconditions and Effects

It is necessary to include in a KB not only facts about the initial situation but also about world dynamics: what actions do.

Actions typically have preconditions: what needs to be true for the action to be performed

- $\mathit{Poss}(\mathit{pickup}(r, x), s) \equiv \forall z. \neg \mathit{Holding}(r, z, s) \wedge \neg \mathit{Heavy}(x) \wedge \mathit{NextTo}(r, x, s)$
  a robot can pickup an object iff it is not holding anything, the object is not too heavy and the robot is next to the object (free vars are univ quant)
- $\mathit{Poss}(\mathit{repair}(r, x), s) \equiv \mathit{HasGlue}(r, s) \wedge \mathit{Broken}(x, s)$
  it is possible to repair an object iff the object is broken and the robot has glue

Actions typically have effects: the fluents that change as the result of performing the action:

- $x \rightarrow \mathit{Broken}(x, \mathit{do}(\mathit{drop}(r, x), s))$
  dropping a fragile object causes it to break
- $\neg \mathit{Broken}(x, \mathit{do}(\mathit{repair}(r, x), s))$
  repairing an object causes it to be unbroken

The Frame Problem in SC

We also need to know which fluents are *unaffected* by actions

- $\mathit{Color}(x, c, s) \rightarrow \mathit{Color}(x, c, \mathit{do}(\mathit{drop}(r, x), s))$
  (dropping an object does not change it's color)
- $\neg \mathit{Broken}(x, s) \wedge [x \neq y \vee \neg \mathit{Fragile}(x)] \rightarrow \neg \mathit{Broken}(x, \mathit{do}(\mathit{drop}(r, y), s))$
  (not breaking things)

These are sometimes called frame axioms **Problem**: need to know a vast number of such axioms (Few actions affect the value of a given fluent; most do nothing to it)

an object's color is unaffected by picking things up, opening a door, using the phone, turning on the light, electing a new president, etc. But it can change after painting, for example.

The frame problem:

- in building KB, need to think of these (about) $2 \times A \times F$ facts about what does not change
- the system needs to reason efficiently with them

What counts as a solution?

Suppose the person responsible for building a KB has written down *all* the effect axioms

> for each fluent F and action A that can cause the truth value of F to change, an axiom of the form $R(s) \rightarrow \pm F(do(A, s))]$,

where $R(s)$ is some condition on s

We want a *systematic* procedure for generating all the frame axioms from these effect axioms

If possible, we also want a *parsimonious* representation for them (since in their simplest form, there are way too many)

Why do we want such as solution?

- frame axioms are necessary to reason about actions and are not entailed by the other axioms
- convenience for the KB builder, for theorizing about actions, and modularity (only add effect axioms, accuracy: no inadvertent omissions)

The projection task

What can we do with the situation calculus?

We will see later that planning is also possible

A simpler job we can handle directly is called the projection task

Given a sequence of actions, determined what would be true in the situation that results from performing that sequence

This can be formalized as follows:

Suppose that $R(s)$ is a formula with a free situation variable s.
To find out if $R(s)$ would be true after performing $\langle a_1, \ldots, a_n \rangle$ in the initial situation, we determine whether or not

$$KB \models R(\textit{do}(a_n, \textit{do}(a_n, \textit{do}(a_{n-1}, \ldots, \textit{do}(a_1, S_0) \ldots)))$$

For example, using the effect and frame axioms from before, it follows that $\neg \textit{Broken}(B, s)$ would hold after doing the sequence

$$\langle \textit{pickup}(A), \textit{pickup}(B), \textit{drop}(B), \textit{repair}(B), \textit{drop}(A) \rangle$$

The legality task

The projection task above asks if a condition would hold after performing a sequence of actions, but not whether that sequence can in fact be properly executed.

We call a situation legal if it is the inititial situation or the result of performing and action whose preconditions are satisfied starting in a legal situation.

The legality task is the task of determining whether a sequence of actions leads to a legal situation.

This can be formalized as follows:

To find out if the sequence $\langle a_1, \ldots a_n \rangle$ can be legally performed in the initial situation, we determine whether or not

$$KB \models \mathit{Poss}(a_i, \ldots, \mathit{do}(a_1, S_0) \ldots))$$

for every i such that $1 \leq i \leq n$

Normal form for effect axioms

Suppose there are two positive effect axioms for the fluent *Broken* :

$\textit{Fragile}(x) \rightarrow \textit{Broken}(x, \textit{do}(\textit{drop}(r, x), s))$

$\textit{NextTo}(b, x, s) \rightarrow \textit{Broken}(x, \textit{do}(\textit{explode}(b), s))$

These can be rewritten as

$\exists r\{a = \textit{drop}(r, x) \wedge \textit{Fragile}(x)\} \vee \exists b\{a = \textit{explode}(b) \wedge \textit{NextTo}(b, x, s)\} \rightarrow \textit{Broken}(x, \textit{do}(a, s))$

Similarly, consider the negative effect axiom:

$\neg \textit{Broken}(x, \textit{do}(\textit{repair}(r, x), s))$

which can be rewritten as

$\exists r\{a = \textit{repair}(r, x)\} \rightarrow \neg \textit{Broken}(x, \textit{do}(a, s))$ In general, for any fluent F, we can rewrite all the effect axioms as two formulas of the form:

$P_F(\mathbf{x}, a, s) \rightarrow F(\mathbf{x}, \textit{do}(a, s))$ (1)

$N_F(\mathbf{x}, a, s) \rightarrow \neg F(\mathbf{x}, \textit{do}(a, s))$ (2)

(both are formulas with free variables which are among x_i, a and s)

Explanation closure

Now make a completeness assumption regarding these effect axioms:
assume that (1) and (2) characterize *all* the conditions under which an action a changes the value of fluent F.

This can be formalized by explanation closure axioms:

$\neg F(\mathbf{x}, s) \wedge F(\mathbf{x}, \mathit{do}(a, s)) \rightarrow P_F(\mathbf{x}, a, s)$ (3)

if F was false and was made true by doing action a then condition P_F must have been true

$F(\mathbf{x}, s) \wedge \neg F(\mathbf{x}, \mathit{do}(a, s)) \rightarrow N_F(\mathbf{x}, a, s)$ (4)

if F was true and was made false by doing action a then condition N_F must have been true

These explanation closure axioms are in fact disguised versions of frame axioms!

$\neg F(\mathbf{x}, s) \wedge \neg P_F(\mathbf{x}, a, s) \rightarrow \neg F(\mathbf{x}, \mathit{do}(a, s))$

$F(\mathbf{x}, s) \wedge \neg N_F(\mathbf{x}, a, s) \rightarrow F(\mathbf{x}, \mathit{do}(a, s))$

Successor state axioms

Further assume that our KB entails the following

a) integrity of the effect axioms: $\neg\exists\mathbf{x}, a, s. P_F(\mathbf{x}, a, s) \wedge N_F(\mathbf{x}, a, s)$

b) unique names for actions:

$$A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \rightarrow (x_1 = y_1) \wedge \ldots \wedge (x_n = y_n)$$

$A(x_1, \ldots, x_n) \neq B(y_1, \ldots, y_n)$ where A and B are distinct.

Then it can be shown that KB entails that (1),(2),(3) and (4) together are logically equivalent to

$$F(\mathbf{x}, \mathit{do}(a, s)) \equiv P_F(\mathbf{x}, a, s) \vee (F(\mathbf{x}, s) \wedge \neg N_F(\mathbf{x}, a, s))$$

This is called the successor state axiom for F.

For example, the successor state axiom for the *Broken* fluent is:

$$\begin{aligned}
&\mathit{Broken}(x, \mathit{do}(a, s)) \equiv \\
&\quad \exists r\{a = \mathit{drop}(r, x) \wedge \mathit{Fragile}(x)\} \\
&\quad \vee\exists b\{a = \mathit{explode}(b) \wedge \mathit{NextTo}(b, x, s)\} \\
&\quad \vee\mathit{Broken}(x, s) \wedge \neg\exists r\{a = \mathit{repair}(r, x)\}
\end{aligned}$$

An object x is broken after doing action a
iff
a is a dropping action and x is fragile
or a is a bomb exploding
(where x is next to the bomb)
or x was already broken and
a is not the action of repairing it

A simple solution to the frame problem

This simple solution to the frame problem was introduced by Ray Reiter, and it yields the following axioms:

- one successor state axiom per fluent
- on precondition axiom per action
- unique name axioms for actions

Moreover, we do not get fewer axioms at the expense of prohibitively long ones. The length of a successor state axiom is roughly proportional to the number of actions which affect the truth value of the fluent
The conciseness and perspicuity of the solution relies on

- quantification over actions
- the assumptions that relatively few actions affect each fluent
- the completeness assumption (for effects)

Moreover, the solution depends on the fact that actions always have *deterministic* effects.

Additional example

Blocks world in situation calculus using clauses

initial state:

$\mathit{On}(a, table, s_0)$

$\mathit{On}(b, c, s_0)$

$\mathit{On}(c, table, s_0)$

$\mathit{Clear}(a, s_0)$

$\mathit{Clear}(b, s_0)$

goal:

$\neg \mathit{On}(a, b, s) \vee \neg \mathit{On}(b, c, s)$

effect axiom:

$\neg \mathit{Clear}(x, s) \vee \neg \mathit{Clear}(y, s) \vee$

$\mathit{On}(x, y, \mathit{do}(\mathit{move}(x, y), s))$

successor state axiom:

$\forall x, y, s, a.$

$\mathit{On}(x, y, \mathit{Do}(a, s)) \leftrightarrow$

$\mathit{On}(x, y, s) \wedge (\forall z. a = \mathit{Move}(x, z) \rightarrow y = z)$

$\vee\ (\mathit{Clear}(x, s) \wedge \mathit{Clear}(y, s) \wedge a = \mathit{Move}(x, y))$

frame axioms as clauses:

$\neg \mathit{On}(x, y, s) \vee a = \mathit{move}(x, Z(x, y, z, s, a)) \vee \mathit{On}(x, y, \mathit{do}(a, s))$

$\neg \mathit{On}(x, y, s) \vee a = \mathit{move}(x, Z(x, y, z, s, a)) \vee \mathit{On}(x, y, \mathit{do}(a, s))$

Using the Situation Calculus

Situation calculus can be used to represent what is known about the state of the world and the available actions.

The planning problem can be formulated as follows:

Given a formula *Goal(s)*, find

a sequence of actions ***a*** such that

$KB \models \textit{Goal}(\textit{do}(\mathbf{a}, S_0)) \wedge \textit{Legal}(\textit{do}(\mathbf{a}, S_0))$

where $\textit{do}(\langle a_1, \ldots, a_n \rangle, S_0)$ is an abbreviation for

$\textit{do}(a_n, \textit{do}(a_{n-1}, \ldots, \textit{do}(a_2, \textit{do}(a_1, S_0)) \ldots))$

and where $\textit{Legal}(\langle a_1, \ldots, a_n \rangle, S_0)$ is an abbreviation for

$\textit{Poss}(a_1, S_0) \wedge \textit{Poss}(a_2, do(a_1, S_0)) \wedge \ldots \wedge \textit{Poss}(a_n, do(\langle a_1, \ldots, a_{n-1} \rangle, S_0))$

So, given a goal formula, we want an action sequence s.t.

that formula holds in in the situation

that results from executing the actions, and

it is possible to execute each action in the appropriate situation

Planning by answer extraction

Having formulated planning this way, we can use resolution with answer extraction to find a sequence of actions:

$$KB \models \exists s. \mathit{Goal}(s) \land \mathit{Legal}(s)$$

We can see how this will work using a simplified version of a previous example:

An object is on the table that we would like to have on the floor. Dropping it will put it on the floor, and we can drop it, provided we are holding it. To hold it, we need to pick it up, and we can always do so.

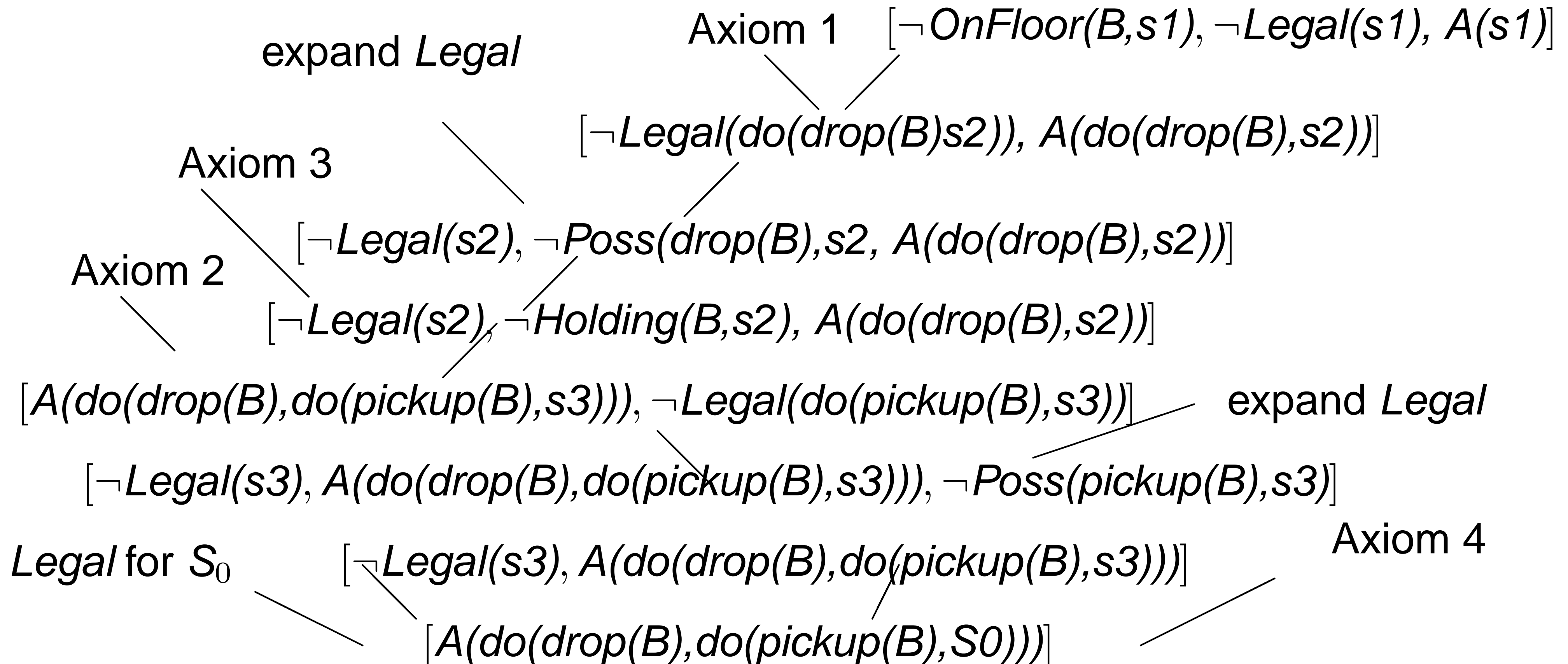
effects: $\mathit{OnFloor}(x, \mathit{do}(\mathit{drop}(x), s))$ and $\mathit{Holding}(x, \mathit{do}(\mathit{pickup}(x), s))$
(note: ignoring frame problem)

preconds: $\mathit{Holding}(x, s) \rightarrow \mathit{Poss}(\mathit{drop}(x), s)$ and $\mathit{Poss}(\mathit{pickup}(x), s)$

initial: $\mathit{OnTable}(B, S_0)$

goal: $\mathit{OnFloor}(B, s)$

Deriving a plan



[plan] initial situation: pickup block B, and in resulting situation, drop B

Prolog implementations

Now, since all of the required elements here can directly be represented using Horn clauses, we can employ Prolog for planning:

```
onfloor(X,do(drop(X),S)).
holding(X,do(pickup(X),S)).
poss(drop(X),S) :- holding(X,S).
poss(pickup(X),S).
ontable(b,s0).
legal(s0).
legal(do(A,S)) :- poss(A,S), legal(S).
```

With the Prolog goal $?-\mathtt{onfloor}(\mathtt{b},\mathtt{S}),\mathtt{legal}(\mathtt{S}).$
we get the solution $\mathtt{S}=\mathtt{do}(\mathtt{drop}(\mathtt{b}),\mathtt{do}(\mathtt{pickup}(\mathtt{b}),\mathtt{s0}))$

Thus, in simple cases, planning can be computed easily. In general, resolution theorem proving in a full first-order setting for planning involves more things.

Planning as theorem proving

Let *Sys* be a logical description of an action domain, and let *Goal* be a goal formula. Planning is very simple to define; just compute a proof for:

$$Sys \models Goal$$

and collect useful structures/substitutions from the proof! This way:

planning = theorem proving

(something that has been known for a long time, starting from Green, Waldinger etc. in the sixties)

In fact, providing useful answer substitutions is the main purpose of Prolog, e.g. the answer $\texttt{X} = \texttt{mary}$ to the query $? - \texttt{parent}(\texttt{X}, \texttt{john})$ is usually more useful than the notion that the query is a logical consequence.

In a similar way, we can equate *probabilistic* planning with theorem proving in probabilistic logic!

Three solutions

We have seen three types of models for change

- the state-operator: potentially needs all frame axioms, requiring $O(FA)$ axioms (F and A are numbers of fluents and actions).
- STRIPS uses CWA + procedural semantics of actions. It only needs to specify $O(A)$ actions, but the rules can become long and tedious.
- Situation calculus requires one axiom *per fluent* and promises that they tend to stay compact (total $O(AE)$ with E the number of effects).

Sitcalc does not represent the state explicitly, i.e. inferring the truth value of a fluent *Goal*(*do*(**a**, S_0)) requires to *reason all the way back to the initial situation* (called: logical regression).

Alternative representations

- Planning languages ADL, PDDL, etc.
- Agent models, BDI, 3APL, AgentSpeak, etc.
- Action logics, A, B, C, ...
- Event Calculus (event recognition, abductive planning)
- Fluent Calculus (flux agent language, constraints)
- Recently: combined action calculus (Thielscher)

Conclusions

- three different models for action and change
- three ways to solve frame problems
- STRIPS and SitCalc representative for many action formalisms
- For practical experience: assignment 2 (forthcoming)
- lots of things *not* in these models (continuous change, probability, utility, control structures, sensing, knowledge and belief, exogeneous change, explicit time, multi-agent actions, etc.)

Next week: control policies (Golog), some probabilistic actions, and vision

Literature

- Required [see blackboard]: Brachman and Levesque (Sections 14.1, 14.2, 15.1, 15.2)
- Required [see blackboard]: Russell and Norvig (2nd edition) (Section 10.3)
- background: Poole and Mackworth (2010) (Section 14.1.1.)