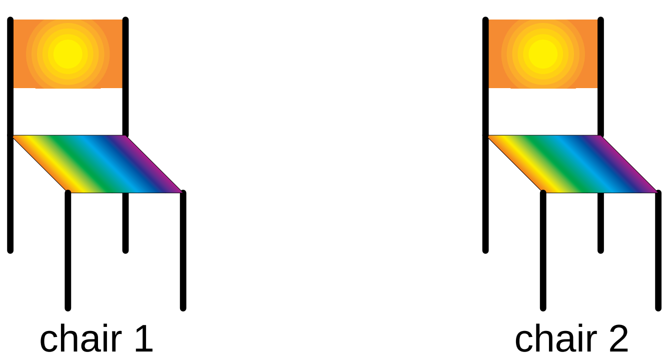
Equality

- Sometimes two terms denote the same individual.
- Example: Clark Kent & superman. 4×4 & $11 + 5$. The projector we used last Friday & this projector.
- Ground term t_1 equals ground term t_2, written $t_1 = t_2$, is true in interpretation I if t_1 and t_2 denote the same individual in interpretation I.

Equality doesn't mean similarity



$chair1 \neq chair2$

$chair\_on\_right = chair2$

$chair\_on\_right$ is not similar to $chair2$, it is $chair2$.

Allowing Equality Assertions

- Without equality assertions, the only thing that is equal to a ground term is itself.

  This can be captured as though you had the assertion $X = X$. Explicit equality never needs to be used.

- If you allow equality assertions, you need to derive what follows from them. Either:
  - axiomatize equality like any other predicate
  - build special-purpose inference machinery for equality

Axiomatizing Equality

$$X = X.$$

$$X = Y \leftarrow Y = X.$$

$$X = Z \leftarrow X = Y \wedge Y = Z.$$

For each n-ary function symbol f there is a rule of the form

$$f(X_1, \ldots, X_n) = f(Y_1, \ldots, Y_n) \leftarrow X_1 = Y_1 \wedge \cdots \wedge X_n = Y_n.$$

For each n-ary predicate symbol p, there is a rule of the form

$$p(X_1, \ldots, X_n) \leftarrow p(Y_1, \ldots, Y_n) \wedge X_1 = Y_1 \wedge \cdots \wedge X_n = Y_n.$$

Special-Purpose Equality Reasoning

paramodulation: if you have $t_1 = t_2$, then you can replace any occurrence of t_1 by t_2.

Treat equality as a **rewrite rule**, substituting equals for equals.

You select a **canonical representation** for each individual and rewrite all other representations into that representation.

Example: treat the sequence of digits as the canonical representation of the number.

Example: use the student number as the canonical representation for students.

Unique Names Assumption

The convention that different ground terms denote different individuals is the **unique names assumption.**

for every pair of distinct ground terms t_1 and t_2, assume $t_1 \neq t_2$, where "$\neq$" means "not equal to."

Example: For each pair of courses, you don't want to have to state, $math302 \neq psyc303$, …

Example: Sometimes the unique names assumption is inappropriate, for example $3 + 7 \neq 2 \times 5$ is wrong.

Axiomatizing Inequality for the UNA

- $c \neq c'$ for any distinct constants c and c'.
- $f(X_1, \ldots, X_n) \neq g(Y_1, \ldots, Y_m)$ for any distinct function symbols f and g.
- $f(X_1, \ldots, X_n) \neq f(Y_1, \ldots, Y_n) \leftarrow X_i \neq Y_i$, for any function symbol f. There are n instances of this schema for every n-ary function symbol f (one for each i such that $1 \leq i \leq n$).
- $f(X_1, \ldots, X_n) \neq c$ for any function symbol f and constant c.
- $t \neq X$ for any term t in which X appears (where t is not the term X).

Top-down procedure and the UNA

- Inequality isn't just another predicate. There are infinitely many answers to $X \neq f(Y)$.
- If you have a subgoal $t_1 \neq t_2$, for terms t_1 and t_2 there are three cases:
  - t_1 and t_2 don't unify. In this case, $t_1 \neq t_2$ succeeds.
  - t_1 and t_2 are identical including having the same variables in the same positions. Here $t_1 \neq t_2$ fails.
  - Otherwise, there are instances of $t_1 \neq t_2$ that succeed and instances of $t_1 \neq t_2$ that fail.

Implementing the UNA

- Recall: in SLD resolution you can select any subgoal in the body of an answer clause to solve next.
- Idea: only select inequality when it will either succeed or fail, otherwise select another subgoal. Thus you are delaying inequality goals.
- If only inequality subgoals remain, and none fail, the query succeeds.

Actions and Planning

- Agents reason in time
- Agents reason about time

Time passes as an agent acts and reasons.

Given a goal, it is useful for an agent to think about what it will do in the future to determine what it will do now.

Representing Time

Time can be modeled in a number of ways:

- **Discrete time** Time can be modeled as jumping from one time point to another.
- **Continuous time** You can model time as being dense.
- **Event-based time** Time steps don't have to be uniform; you can consider the time steps between interesting events.
- **State space** Instead of considering time explicitly, you can consider actions as mapping from one state to another.

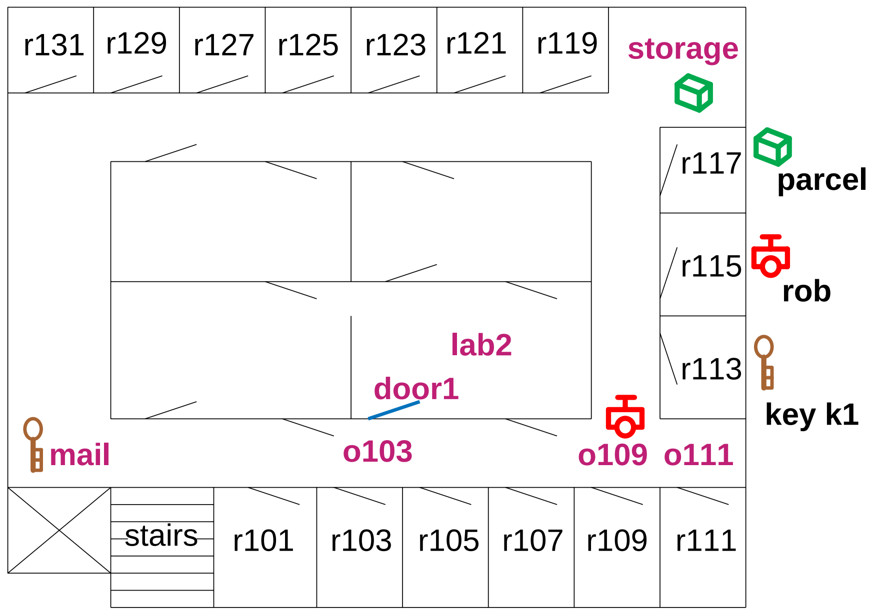
You can model time in terms of **points** or **intervals.**

Time and Relations

When modeling relations, you distinguish two basic types:

- **Static relations** are those relations whose value does not depend on time.
- **Dynamic relations** are relations whose truth values depends on time. Either
  - **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.

The Delivery Robot World



Modeling the Delivery Robot World

Individuals: rooms, doors, keys, parcels, and the robot.

Actions:

- move from room to room
- pick up and put down keys and packages
- unlock doors (with the appropriate keys)

Relations: represent

- the robot's position
- the position of packages and keys and locked doors
- what the robot is holding

Example Relations

- $at(Obj, Loc)$ is true in a situation if object *Obj* is at location *Loc* in the situation.
- $carrying(Ag, Obj)$ is true in a situation if agent *Ag* is carrying *Obj* in that situation.
- $sitting\_at(Obj, Loc)$ is true in a situation if object *Obj* is sitting on the ground (not being carried) at location *Loc* in the situation.
- $unlocked(Door)$ is true in a situation if door *Door* is unlocked in the situation.
- $autonomous(Ag)$ is true if agent *Ag* can move

autonomously. This is static.

- $opens(Key, Door)$ is true if key *Key* opens door *Door*. This is static.
- $adjacent(Pos_1, Pos_2)$ is true if position Pos_1 is adjacent to position Pos_2 so that the robot can move from Pos_1 to Pos_2 in one step.
- $between(Door, Pos_1, Pos_2)$ is true if *Door* is between position Pos_1 and position Pos_2. If the door is unlocked, the two positions are adjacent.

Actions

- *move*(*Ag*, *From*, *To*): agent *Ag* moves from location *From* to adjacent location *To*. The agent must be sitting at location *From*.
- *pickup*(*Ag*, *Obj*) agent *Ag* picks up *Obj*. The agent must be at the location that *Obj* is sitting.
- *putdown*(*Ag*, *Obj*) the agent *Ag* puts down *Obj*. It must be holding *Obj*.
- *unlock*(*Ag*, *Door*) agent *Ag* unlocks *Door*. It must be outside the door and carrying the key to the door.

Initial Situation

sitting_at(*rob*, *o*109).

sitting_at(*parcel*, *storage*).

sitting_at(*k*1, *mail*).

Static Facts

between(*door*1, *o*103, *lab*2).

opens(*k*1, *door*1).

autonomous(*rob*).

Derived Relations

$at(Obj, Pos) \leftarrow sitting\_at(Obj, Pos).$

$at(Obj, Pos) \leftarrow carrying(Ag, Obj) \wedge at(Ag, Pos).$

$adjacent(o109, o103).$

$adjacent(o103, o109).$

$\ldots$

$adjacent(lab2, o109).$

$adjacent(P_1, P_2) \leftarrow$
$\quad between(Door, P_1, P_2) \wedge$
$\quad unlocked(Door).$

STRIPS Representation

- State-based view of time.
- The actions are external to the logic.
- Given a state and an action, the STRIPS representation is used to determine
  - whether the action can be carried out in the state
  - what is true in the resulting state

STRIPS Representation: Idea

- Predicates are primitive or derived.
- Use normal rules for derived predicates.
- The STRIPS representation is used to determine the truth values of primitive predicates based on the previous state and the action.
- Based on the idea that most predicates are unaffected by a single action.
- STRIPS assumption: Primitive relations not mentioned in the description of the action stay unchanged.

STRIPS Representation of an action

The STRIPS representation for an action consists of:

preconditions A list of atoms that need to be true for the action to occur

delete list A list of those primitive relations no longer true after the action

add list A list of the primitive relations made true by the action

STRIPS Representation of “pickup”

The action $pickup(Ag, Obj)$ can be defined by:

- preconditions $[autonomous(Ag), Ag \neq Obj, at(Ag, Pos), sitting\_at(Obj, Pos)]$
- delete list $[sitting\_at(Obj, Pos)]$
- add list $[carrying(Ag, Obj)]$

STRIPS Representation of “move”

The action $move(Ag, Pos_1, Pos_2)$ can be defined by:

- preconditions $[autonomous(Ag), adjacent(Pos_1, Pos_2, S)\ , sitting\_at(Ag, Pos_1)]$
- delete list $[sitting\_at(Ag, Pos_1)]$
- add list $[sitting\_at(Ag, Pos_2)]$

Example Transitions

$$\left\{\begin{array}{l} sitting\_at(rob, o109). \\ sitting\_at(parcel, storage). \\ sitting\_at(k1, mail). \end{array}\right\}$$

$$move(rob, o109, storage) \longrightarrow \left\{\begin{array}{l} sitting\_at(rob, storage). \\ sitting\_at(parcel, storage). \\ sitting\_at(k1, mail). \end{array}\right\}$$

$$pickup(rob, parcel) \longrightarrow \left\{\begin{array}{l} sitting\_at(rob, storage). \\ carrying(rob, parcel). \\ sitting\_at(k1, mail). \end{array}\right\}$$

Situation Calculus

- State-based representation where the states are denoted by terms.
- A situation is a term that dentotes a state.
- There are two ways to refer to states:
  - *init* denotes the initial state
  - $do(A, S)$ denotes the state resulting from doing action A in state S, if it is possible to do A in S.
- A situation also encodes how to get to the state it denotes.

Example States

- *init*
- $do(move(rob, o109, o103), init)$
- $do(move(rob, o103, mail),$
  $do(move(rob, o109, o103),$
  $init)).$
- $do(pickup(rob, k1),$
  $do(move(rob, o103, mail),$
  $do(move(rob, o109, o103),$
  $init))).$

Using the Situation Terms

- Add an extra term to each dynamic predicate indicating the situation.
- Example Atoms:

  $at(rob, o109, init)$

  $at(rob, o103, do(move(rob, o109, o103), init))$

  $at(k1, mail, do(move(rob, o109, o103), init))$

Axiomatizing using the Situation Calculus

- You specify what is true in the initial state using axioms with *init* as the situation parameter.
- Primitive relations are axiomatized by specifying what is true in situation $do(A, S)$ in terms of what holds in situation S.
- Derived relations are defined using clauses with a free variable in the situation argument.
- Static relations are defined without reference to the situation.

Initial Situation

$sitting\_at(rob, o109, init).$

$sitting\_at(parcel, storage, init).$

$sitting\_at(k1, mail, init).$

Derived Relations

$adjacent(P_1, P_2, S) \leftarrow$

$\quad between(Door, P_1, P_2) \wedge$

$\quad unlocked(Door, S).$

$adjacent(lab2, o109, S).$

...

When are actions possible?

$poss(A, S)$ is true if action A is possible in state S.

$$poss(putdown(Ag, Obj), S) \leftarrow$$
$$carrying(Ag, Obj, S).$$

$$poss(move(Ag, Pos_1, Pos_2), S) \leftarrow$$
$$autonomous(Ag) \wedge$$
$$adjacent(Pos_1, Pos_2, S) \wedge$$
$$sitting\_at(Ag, Pos_1, S).$$

Axiomatizing Primitive Relations

Example: Unlocking the door makes the door unlocked:

$$unlocked(Door, do(unlock(Ag, Door), S)) \leftarrow$$
$$poss(unlock(Ag, Door), S).$$

Frame Axiom: No actions lock the door:

$$unlocked(Door, do(A, S)) \leftarrow$$
$$unlocked(Door, S) \wedge$$
$$poss(A, S).$$

Example: axiomatizing *carried*

Picking up an object causes it to be carried:

$$carrying(Ag, Obj, do(pickup(Ag, Obj), S)) \leftarrow$$
$$poss(pickup(Ag, Obj), S).$$

Frame Axiom: The object is being carried if it was being carried before unless the action was to put down the object:

$$carrying(Ag, Obj, do(A, S)) \leftarrow$$
$$carrying(Ag, Obj, S) \wedge$$
$$poss(A, S) \wedge$$
$$A \neq putdown(Ag, Obj).$$

More General Frame Axioms

The only actions that undo *sitting_at* for object *Obj* is when *Obj* moves somewhere or when someone is picking up *Obj*.

$$sitting\_at(Obj, Pos, do(A, S)) \leftarrow$$
$$poss(A, S) \wedge$$
$$sitting\_at(Obj, Pos, S) \wedge$$
$$\forall Pos_1 \; A \neq move(Obj, Pos, Pos_1) \wedge$$
$$\forall Ag \; A \neq pickup(Ag, Obj).$$

The last line is equivalent to:

$$\sim\exists Ag \, A = pickup(Ag, Obj)$$

which can be implemented as

$$sitting\_at(Obj, Pos, do(A, S)) \leftarrow$$
$$\cdots \wedge \cdots \wedge \cdots \wedge$$
$$\sim is\_pickup\_action(A, Obj).$$

with the clause:

$$is\_pickup\_action(A, Obj) \leftarrow$$
$$A = pickup(Ag, Obj).$$

which is equivalent to:

$$is\_pickup\_action(pickup(Ag, Obj), Obj).$$

STRIPS and the Situation Calculus

- Anything that can be stated in STRIPS can be stated in the situation calculus.
- The situation calculus is more powerful. For example, the "drop everything" action.
- To axiomatize STRIPS in the situation calculus, we can use $\boxed{holds(C, S)}$ to mean that C is true in situation S.

$holds(C, do(A, W)) \leftarrow$	
$\quad preconditions(A, P) \wedge$	The preconditions of
$\quad holdsall(P, W) \wedge$	of A all hold in W.
$\quad add\_list(A, AL) \wedge$	C is on the
$\quad member(C, AL).$	addlist of A.
$holds(C, do(A, W)) \leftarrow$	
$\quad preconditions(A, P) \wedge$	The preconditions of
$\quad holdsall(P, W) \wedge$	of A all hold in W.
$\quad delete\_list(A, DL) \wedge$	C isn't on the
$\quad notin(C, DL) \wedge$	deletelist of A.
$\quad holds(C, W).$	C held before A.