CSE 473

Automated Planning

Dan Weld

(With slides by UW AI faculty & Dana Nau)

I have a plan - a plan that cannot possibly fail.

- Inspector Clousseau

Logistics

- HW1 due in one week (Fri 5/4)
  - Parts due in between:
    - Monday draft answer to problem 1
    - Wed give feedback on another person's answer

Overview

- Introduction & Agents
- Search, Heuristics & CSPs
- Adversarial Search
- **Logical Knowledge Representation**
- **Planning** & MDPs
- Reinforcement Learning
- Uncertainty & Bayesian Networks
- Machine Learning
- NLP & Special Topics

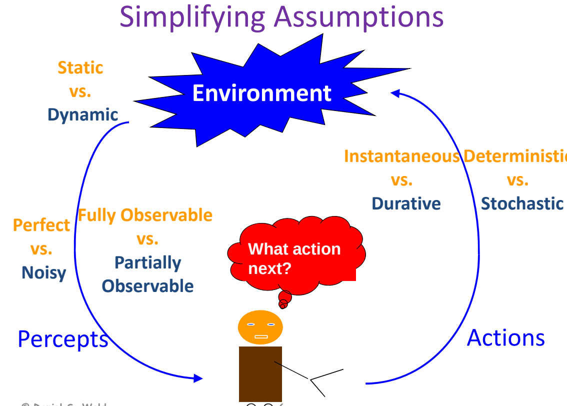
Today's Topics

- Logic for specifying planning domains
- Planning graph for computing heuristics
- Compiling planning to SAT

Planning

- Given
  - a logical description of the **initial situation**,
  - a logical description of the **goal conditions**, and
  - a logical description of a set of **possible actions**,
- Find
  - a **sequence of actions** (a **plan of actions**) that brings us from the initial situation to a situation in which the goal conditions hold.

Simplifying Assumptions



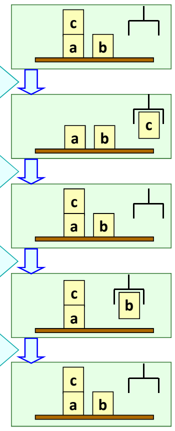
Classical Operators

unstack(*x*,*y*)
  Precond: on(*x*,*y*), clear(*x*), handempty
  Effects: ¬on(*x*,*y*), ¬clear(*x*), ¬handempty, holding(*x*), clear(*y*)

stack(*x*,*y*)
  Precond: holding(*x*), clear(*y*)
  Effects: ¬holding(*x*), ¬clear(*y*), on(*x*,*y*), clear(*x*), handempty

pickup(*x*)
  Precond: ontable(*x*), clear(*x*), handempty
  Effects: ¬ontable(*x*), ¬clear(*x*), ¬handempty, holding(*x*)

putdown(*x*)
  Precond: holding(*x*)
  Effects: ¬holding(*x*), ontable(*x*), clear(*x*), handempty



Driving

Static facts

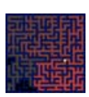
```
(defschema (drive)
   :parameters    (?v ?s ?d)
   :precondition  (and (vehicle ?v) (at ?v ?s)
                       (location ?s) (location ?d)
                       (road-connected ?s ?d))
   :effect        (and (at ?v ?d) (not (at ?v ?s))
                       (forall (object ?o)
                          (when (in ?o ?v)
                                (and (at ?o ?d)) (not (at ?o ?s))))))
```

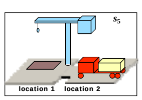
Universally quantified conditional schemata for driving.

Planning *vs.* Problem-Solving ?

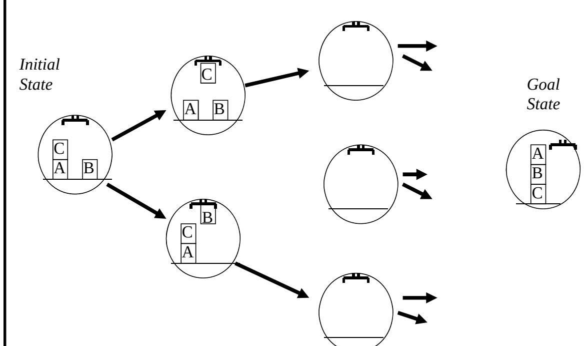
Basic difference: **Explicit, logic-based representation**

- States/Situations: descriptions of the world by logical formulae
  → agent can explicitly reason about the world.
- Goal conditions as logical formulae vs. goal test (black box)
  → agent can reflect on its goals.
- Operators/Actions: Transformations on logical formulae
  → agent can reason about the effects of actions by inspecting the definition of its operators.





Forward World-Space Search



Heuristics for State-Space Search

- Count number of false goal propositions in current state
  Admissible?
  NO
- Subgoal independence assumption:
  – Cost of solving conjunction is sum of cost of solving each subgoal independently
  – Optimistic: ignores negative interactions
  – Pessimistic: ignores redundancy
  – Admissible? No
  – Can you make this admissible?

Heuristic Generation II

unstack(*x*,*y*)
  Precond: ~~on(*x*,*y*), clear(*x*), handempty~~
  Effects: ¬on(*x*,*y*), ¬clear(*x*), ¬handempty, holding(*x*), clear(*y*)

stack(*x*,*y*)
  Precond: ~~holding(*x*), clear(*y*)~~
  Effects: ¬holding(*x*), ¬clear(*y*), on(*x*,*y*), clear(*x*), handempty

pickup(*x*)
  Precond: ~~ontable(*x*), clear(*x*), handempty~~
  Effects: ¬ontable(*x*), ¬clear(*x*), ¬handempty, holding(*x*)

putdown(*x*)
  Precond: ~~holding(*x*)~~
  Effects: ¬holding(*x*), ontable(*x*), clear(*x*), handempty

Delete preconditions

Solve relaxed planning problem

Admissable?

Heuristic Generation III

unstack(*x*,*y*)
Precond: on(*x*,*y*), clear(*x*), handempty
Effects: ~~on(*x*,*y*), clear(*x*), handempty,~~ holding(*x*), clear(*y*)

stack(*x*,*y*)
Precond: holding(*x*), clear(*y*)
Effects: ~~holding(*x*), clear(*y*),~~ on(*x*,*y*), clear(*x*), handempty

pickup(*x*)
Precond: ontable(*x*), clear(*x*), handempty
Effects: ~~ontable(*x*), clear(*x*), handempty,~~ holding(*x*)

putdown(*x*)
Precond: holding(*x*)
Effects: ~~holding(*x*)~~, ontable(*x*), clear(*x*), handempty

Delete negative effects

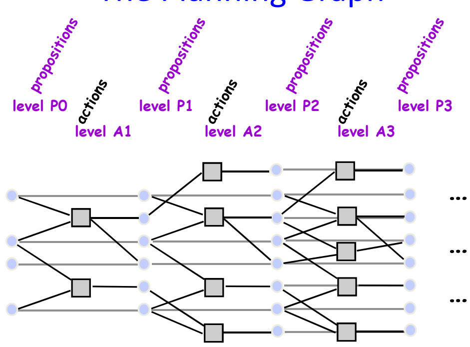
Solve relaxed planning problem

Admissable?

Planning Graph: Basic idea

- Construct a planning graph: encodes constraints on possible plans
- Use this planning graph to compute an informative heuristic (Forward A*)
- Planning graph can be built for each problem in polynomial time

The Planning Graph



Note: a few noops missing for clarity

PG Example

Init(Have(Cake))

Goal(Have(Cake) ∧ Eaten(Cake))

Action(Eat(Cake),
PRECOND: Have(Cake)
EFFECT: ¬Have(Cake) ∧ Eaten(Cake))

Action(Bake(Cake),
PRECOND: ¬ Have(Cake)
EFFECT: Have(Cake))

PG Example

S_0 A_0 S_1

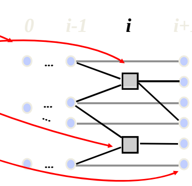
Have(Cake)

¬Eaten(Cake)

Create level 0 from initial problem state.

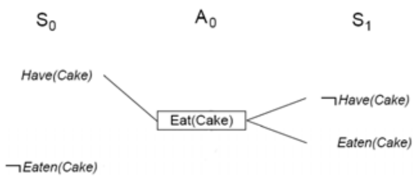
Graph Expansion

Proposition level 0
initial conditions
Action level i
no-op for each proposition at level i-1
action for each operator instance whose preconditions exist at level i-1
Proposition level i
effects of each no-op and action at level i



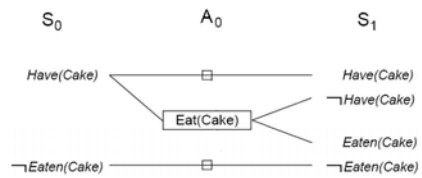
No-op-action(P),
PRECOND: P
EFFECT: P
Have a no-op action for each ground fact

PG Example



Add all applicable actions.
Add all effects to the next state.

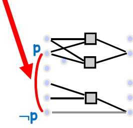
PG Example



Add *persistence actions* (aka no-ops) to map all literals in state S_i to state S_{i+1}.

Mutual Exclusion

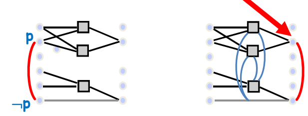
Two propositions are mutex if
- one is the negation of the other



24

Mutual Exclusion

Two proposition are mutex if
- one is the negation of the other
- all ways of achieving them are mutex

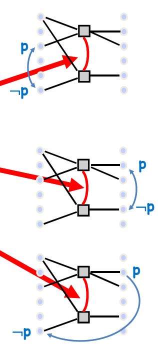


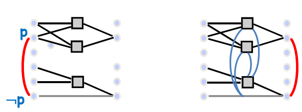
25

Mutual Exclusion

Two actions are mutex if
- they have mutex preconditions
- one clobbers the other's preconditions or effects

Two proposition are mutex if
- one is the negation of the other
- all ways of achieving them are mutex



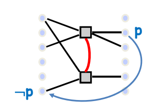


26

Mutual Exclusion

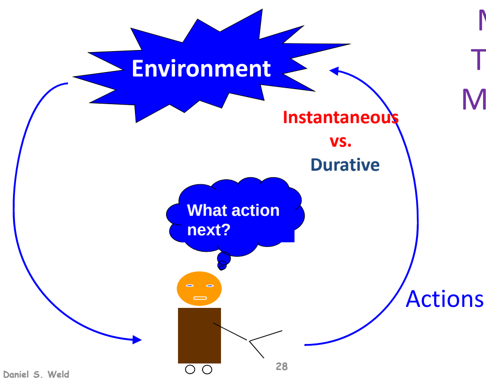
Light-fuse(*match, bomb*)
Precond: lit(*match*), holding(*bomb*)
Effects: will-explode(*bomb*)

Extinguish(*match*)
Precond: lit(*match*)
Effects: ¬lit(*match*)

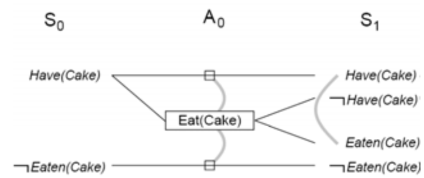


If Result of N>1 Actions is Ambiguous…

Mark Them Mutex

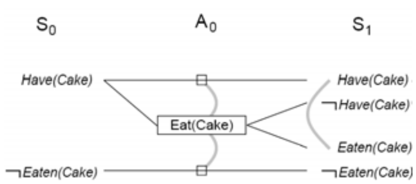


PG Example



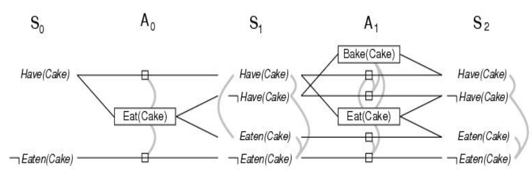
Identify *mutual exclusions* between actions and literals based on potential conflicts.

Cake example

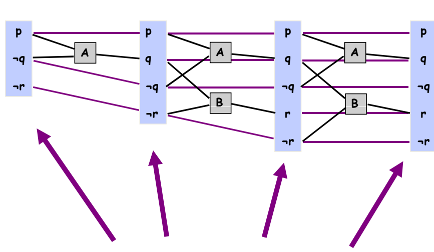


- Level S_1 contains all literals that might result from picking any subset of actions in A_0
  - Conflicts between literals that can't occur together (as a consequence of selection action) are represented by mutex links.
  - S1 defines ***multiple states*** and the mutex links are the constraints that define this set of states.

Cake example

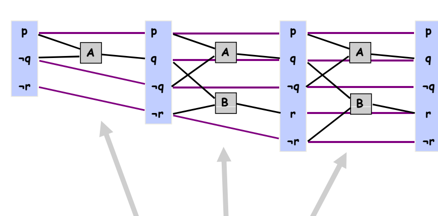


Observation 1



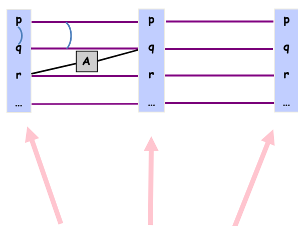
Literals monotonically increase
(always carried forward by no-ops)

Observation 2



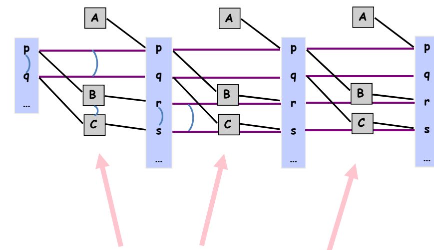
Actions monotonically increase

Observation 3



mutex relationships between literals monotonically decrease

Observation 4



Mutex relationships between actions monotonically decrease

Observation 5

Planning Graph 'levels off'.

- After some time k, all levels are identical
  - Because it's a finite space & montonicity

Properties of Planning Graph

- If goal is absent from last level?
  - Then goal cannot be achieved!
- If there exists a plan to achieve goal?
  - Then goal is present in the last level &
  - No mutexes between conjuncts
- If goal is present in last level (w/ no mutexes) ?
  - There still may not exist any viable plan

Heuristics based on Planning Graph

- Construct planning graph starting from s
- h(s) = level at which goal appears non-mutex
  - Admissible?
  - YES

Planning Graph is Optimistic

Suppose you want to prepare a surprise dinner for your sleeping sweetheart

s_0 = {garbage, cleanHands, quiet}

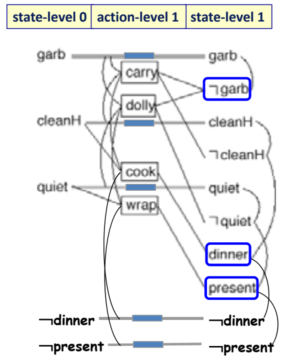
g = {dinner, present, ¬garbage}

Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	*none*	¬garbage, ¬cleanHands
dolly()	*none*	¬garbage, ¬quiet

Also have persistence actions: one for each literal

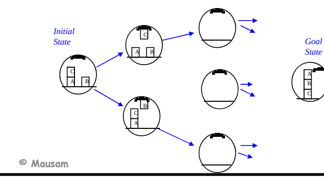
Example (continued)

- Recall the goal is
  - {¬*garbage, dinner, present*}
- Note that in state-level 1,
  - All of them are there
  - None are mutex with each other
- Thus, there's a chance that a plan exists
  - But no actual plan ***does…***
  - Pairwise, the goals are consistent
  - But no consistent way to achieve all three
  - Planning graph ~ k consistency
    But with no fixed limit on k



Fast-Forward (FF)

- Fastest classical planner ~2009
- State space local search
  - Guided by relaxed planning graph
  - Full best-first search to escape plateaus
  - A few other bells and whistles…

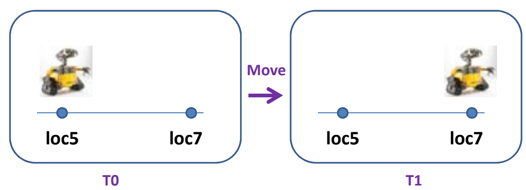


Today's Topics

- Logic for specifying planning domains
- Planning graph for computing heuristics
- Compiling planning to SAT

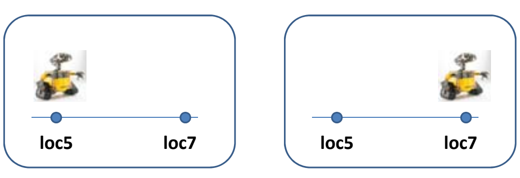
Fluent

- Ground literal whose truth value may change over time
- Eg, at(robbie, location5)
- Not robot(robbie)



Encoding Fluents in Logic

- at(robbie, location7, time1)



Encoding Initial Conditions & Goals

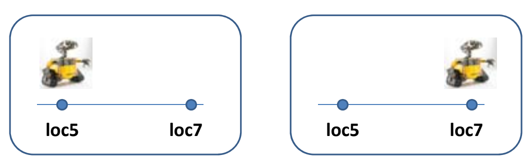
Suppose we only cared about 1-step plans

State desired end conditions:

Φ = at(robbie, loc5, time0) $\wedge$ at(robbie, loc7, time1)

Is Φ satisfiable?

Is P$\wedge$ Q satisfiable?



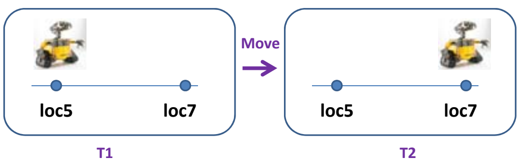
Encoding Action Effects in Logic

move(r, $l1$, $l2$)
Precond: robot(r), at(r,l1), …
Effects: at(r,l2)

$\forall$r, l1, l2, t   move(r,l1,l2,t) => at(r, l2, t+1)

$\forall$r, l1, l2, t   move(r,l1,l2,t) => at(r, l, t)

$\neg$move(r,l1,l2,t) $\vee$ at(r, l, t)



Compiling to Propositional Logic

move(r, $l1$, $l2$)
Precond: robot(r), at(r,l1), …
Effects: at(r,l2)

$\forall$r, l1, l2, t   move(r,l1,l2,t) => at(r, l2, t+1)

Infinite worlds: impossible

But suppose only 2 robots (robbie, sue), 2 locations, 1 action time

move(robbie,loc5,loc7,1) => at(robbie, loc7, 2) $\wedge$
move(robbie,loc7,loc5,1) => at(robbie, loc5, 2) $\wedge$
move(sue,loc5,loc7,1) => at(sue, loc7, 2) $\wedge$
move(sue,loc7,loc5,1) => at(sue, loc5, 2)

Overall Approach

- A *bounded planning problem* is a pair (P,n):
  - P is a planning problem; n is a positive integer
  - Any solution for P of length n is a solution for (P,n)
- Planning algorithm:
- Do iterative deepening like we did with Graphplan:
  - for n = 0, 1, 2, …,
    - encode (P,n) as a satisfiability problem Φ
    - if Φ is satisfiable, then
      - From the set of truth values that satisfies Φ, a solution plan can be constructed, so return it and exit

Encoding Planning Problems

- Encode (P,n) as a formula Φ such that
  $\pi = \langle a_0, a_1, \ldots, a_{n-1}\rangle$ is a solution for (P,n) if and only if
  Φ can be satisfied in a way that makes the fluents $a_0, \ldots, a_{n-1}$ true
- Let
  - A = {all actions in the planning domain}
  - S = {all states in the planning domain}
  - L = {all literals in the language}
- Φ is the conjunct of many other formulas …

Formulas in Φ

- Formula describing the initial state:
  $\bigwedge\{l_0 \mid l \in s_0\} \wedge \bigwedge\{\neg l_0 \mid l \in L - s_0\}$
- Formula describing the goal:
  $\bigwedge\{l_n \mid l \in g^+\} \wedge \bigwedge\{\neg l_n \mid l \in g^-\}$
- For every action a in A, formulas describing what changes a would make if it were the i'th step of the plan:
  - $a_i \Rightarrow \bigwedge\{p_i \mid p \in \text{Precond}(a)\} \wedge \bigwedge\{e_{i+1} \mid e \in \text{Effects}(a)\}$
- *Complete exclusion* axiom:
  - For all actions a and b, formulas saying they can't occur at the same time
    $\neg a_i \vee \neg b_i$
  - this guarantees there can be only one action at a time
- Is this enough?

Frame Axioms

- *Frame axioms*:
  - Formulas describing what *doesn't* change between steps i and i+1
- Several ways to write these
- One way: *explanatory frame axioms*
  - One axiom for every literal l
  - Says that if l changes between s_i and s_{i+1}, then the action at step i must be responsible:
    $(\neg l_i \wedge l_{i+1} \Rightarrow \bigvee_{a \text{ in } A}\{a_i \mid l \in \text{effects}^+(a)\})$
    $\wedge\ (l_i \wedge \neg l_{i+1} \Rightarrow \bigvee_{a \text{ in } A}\{a_i \mid l \in \text{effects}^-(a)\})$

Example

- Planning domain:
  - one robot r1
  - two adjacent locations l1, l2
  - one operator (move the robot)
- Encode (*P*,*n*) where *n* = 1
  - Initial state: {at(r1,l1)}
    Encoding: at(r1,l1,0) ∧ ¬at(r1,l2,0)
  - Goal: {at(r1,l2)}
    Encoding: at(r1,l2,1) ∧ ¬at(r1,l1,1)
  - Operator: see next slide

Example (continued)

- Operator: move(r,l,l')
  precond: at(r,l)
  effects: at(r,l'), ¬at(r,l)

Encoding:

move(r1,l1,l2,0) ⇒ at(r1,l1,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l1,1)
move(r1,l2,l1,0) ⇒ at(r1,l2,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l2,1)
move(r1,l1,l1,0) ⇒ at(r1,l1,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l1,1) } **contradictions**
move(r1,l2,l2,0) ⇒ at(r1,l2,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l2,1) } **(easy to detect)**
move(l1,r1,l2,0) ⇒ …
move(l2,l1,r1,0) ⇒ …
move(l1,l2,r1,0) ⇒ …
move(l2,l1,r1,0) ⇒ … } **nonsensical**

- How to avoid generating the last four actions?
  - Assign data types to the constant symbols
    like we did for state-variable representation

Example (continued)

- Locations: l1, l2
- Robots: r1
- Operator: move(r : robot, l : location, l' : location)
  precond: at(r,l)
  effects: at(r,l'), ¬at(r,l)

Encoding:

move(r1,l1,l2,0) ⇒ at(r1,l1,0) ∧ at(r1,l2,1) ∧ ¬at(r1,l1,1)
move(r1,l2,l1,0) ⇒ at(r1,l2,0) ∧ at(r1,l1,1) ∧ ¬at(r1,l2,1)

Example (continued)

- Complete-exclusion axiom:
  ¬move(r1,l1,l2,0) ∨ ¬move(r1,l2,l1,0)
- Explanatory frame axioms:
  ¬at(r1,l1,0) ∧ at(r1,l1,1) ⇒ move(r1,l2,l1,0)
  ¬at(r1,l2,0) ∧ at(r1,l2,1) ⇒ move(r1,l1,l2,0)
  at(r1,l1,0) ∧ ¬at(r1,l1,1) ⇒ move(r1,l1,l2,0)
  at(r1,l2,0) ∧ ¬at(r1,l2,1) ⇒ move(r1,l2,l1,0)

Extracting a Plan

- Suppose we find an assignment of truth values that satisfies Φ.
  - This means *P* has a solution of length *n*
- For *i*=1,…,*n*, there will be exactly one action *a* such that a_i = *true*
  - This is the *i*'th action of the plan.
- Example (from the previous slides):
  - Φ can be satisfied with move(r1,l1,l2,0) = *true*
  - Thus ⟨move(r1,l1,l2,0)⟩ is a solution for (*P*,0)
    - It's the only solution - no other way to satisfy Φ

Planning

- How to find an assignment of truth values that satisfies Φ?
  - Use a satisfiability algorithm
- Example: the *Davis-Putnam* algorithm
  - First need to put Φ into conjunctive normal form
    e.g., Φ = D ∧ (¬D ∨ A ∨ ¬B) ∧ (¬D ∨ ¬A ∨ ¬B) ∧ (¬D ∨ ¬A ∨ B) ∧ A
  - Write Φ as a set of *clauses* (disjuncts of literals)
    Φ = {{D}, {¬D, A, ¬B}, {¬D, ¬A, ¬B}, {¬D, ¬A, B}, {A}}
  - Two special cases:
    - If Φ = ∅ then Φ is always *true*
    - If Φ = {…, ∅, …} then Φ is always *false* (hence unsatisfiable)