CS 2710 / ISSP 2610

Other Classical Planning Approaches

Planning

- What is classical planning?
- Approaches
  - STRIPS/PDDL
  - State-Space Search
  - Planning Graphs
  - *Situation Calculus*
  - *Partially Ordered Plans*
  - *Satisfiability*

Situation calculus planning

- Intuition: Represent the planning problem using first-order logic
  - Situation calculus lets us reason about changes in the world
  - Use theorem proving to "prove" that a particular sequence of actions, when applied to the situation characterizing the world state, will lead to a desired result

Motivation

- Recall problems with propositional logic. So FOL?
- The robot is in the kitchen.
  - in(robot,kitchen)
- It walks into the living room.
  - in(robot,livingRoom)
- Ooops…
- in(robot,kitchen,2:02pm)
- in(robot,livingRoom,2:17pm)
- But what if you are not sure when it was?
- We can do something simpler than rely on time stamps…

Representation of Time

- Lots of other approaches besides situations, e.g.
  - Temporal Logic, Dynamic Logic
  - Maintaining Knowledge about Temporal Intervals

Situation Calculus

- Logic for reasoning about changes in the state of the world
- The world is described by
  - Sequences of situations of the current state
  - Changes from one situation to another are caused by actions
- The situation calculus allows us to
  - Describe the initial state and a goal state
  - Build the KB that describes the effect of actions (operators)
  - Prove that the KB and the initial state lead to a goal state
  - Extracts a plan as side-effect of the proof

Situation Calculus Ontology

- Actions: terms, such as “forward” and “turn(right))”
- Situations: terms; initial situation s0 and all situations that are generated by applying an action to a situation. result(a,s) names the situation resulting when action a is done in situation s.

Situation Calculus Ontology continued

- Fluents: functions and predicates that vary from one situation to the next. By convention, the situation is the last argument of the fluent.
  ~holding(robot,gold,s0)
- Atemporal or eternal predicates and functions do not change from situation to situation. gold(g1).
  lastName(wumpus,smith).
  adjacent(livingRoom,kitchen).

Modified Wumpus World

- Won't worry about agent's orientation
- Fluent predicates: at(O,X,S) and holding(O,S)
- Initial situation: at(agent,[1,1],s0) ^ at(g1,[1,2],s0)
- But we want to exclude possibilities from the initial situation too…

Initial KB

- All O,X at(O,X,s0) ←→ [O=agent ^ X = [1,1]) v (O=g1 ^ X = [1,2])]
- All O ~holding(O,s0)
- Eternals:
  - gold(g1) ^ adjacent([1,1],[1,2]) ^ adjacent([1,2],[1,1]).

Goal: g1 is in [1,1]

At(g1,[1,1],resultSeq(
[go([1,1],[1,2]),grab(g1),go([1,2],[1,1])],
s0)

Or, planning by answering the query:

Exists S at(g1,[1,1],resultSeq(S,s0))

So, what has to go in the KB for such queries to be answered?...

Axioms for our Wumpus World

- For brevity: we will omit universal quantifiers that range over entire sentence. S ranges over situations, A ranges over actions, O over objects (including agents), G over gold, and X,Y,Z over locations.

Possibility and Effect Axioms

- Possibility axioms:
  - Preconditions → poss(A,S)

- Effect axioms:
  - poss(A,S) → changes that result from that action

Possibility Axioms

- The possibility axioms that an agent can
  - go between adjacent locations,
  - grab a piece of gold in the current location, and
  - release gold it is holding
    - Holding(g,s) => Poss(Release(g),s)

Effect Axioms

- If an action is possible, then certain fluents will hold in the situation that results from executing the action
  - Going from X to Y results in being at Y
  - Grabbing the gold results in holding the gold
  - Releasing the gold results in not holding it
    - Poss(Release(g),s)=>~Holding(g,Result(Release(g),s))

Frame Problem

- We run into the frame problem
- Effect axioms say what changes, but don’t say what stays the same
- A real problem, because (in a non-toy domain), each action affects only a tiny fraction of all fluents

Frame Problem (continued)

- One solution approach is writing explicit frame axioms, such as:

At(O,X,S) ^ ~(O=agent) ^ ~holding(O,S) → at(O,X,result(Go(Y,Z),S))

Representational Frame Problem

- What stays the same?
- A actions, F fluents, and E effects/action (worst case). Typically, E << F
- Want O(AE) versus O(AF) solution

Solving the Representational Frame Problem

- Instead of writing the effects of each action, consider how each fluent predicate evolves over time
- Successor-state axioms:
- Action is possible →

  (fluent is true in result state ←→

  action's effect made it true v

  it was true before and action left it alone)

Blocks world example

- A situation calculus rule for the blocks world:
  - Clear (X, Result(A,S)) $\leftrightarrow$
    [Clear (X, S) $\wedge$
    ($\neg$(A=Stack(Y,X) $\vee$ A=Pickup(X))
    $\vee$ (A=Stack(Y,X) $\wedge$ $\neg$(holding(Y,S))
    $\vee$ (A=Pickup(X) $\wedge$ $\neg$(handempty(S) $\wedge$ ontable(X,S) $\wedge$ clear(X,S))))]
    $\vee$ [A=Stack(X,Y) $\wedge$ holding(X,S) $\wedge$ clear(Y,S)]
    $\vee$ [A=Unstack(Y,X) $\wedge$ on(Y,X,S) $\wedge$ clear(Y,S) $\wedge$ handempty(S)]
    $\vee$ [A=Putdown(X) $\wedge$ holding(X,S)]
- English translation: A block is clear if (a) in the previous state it was clear and we didn't pick it up or stack something on it successfully, or (b) we stacked it on something else successfully, or (c) something was on it that we unstacked successfully, or (d) we were holding it and we put it down.
- Whew!!! There's gotta be a better way!

Situation calculus planning: Analysis

- This is fine in theory, but remember that problem solving (search) is exponential in the worst case
- Also, resolution theorem proving only finds *a* proof (plan), not necessarily a good plan
- So we restrict the language and use a special-purpose algorithm (a planner) rather than general theorem prover

Plan-space planning

- An alternative is to **search through the space of *plans***, rather than situations.
- Start from a **partial plan** which is expanded and refined until a complete plan that solves the problem is generated.
- **Refinement operators** add constraints to the partial plan and modification operators for other changes.
- We can still use STRIPS-style operators:

  Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)

  Op(ACTION: RightSock, EFFECT: RightSockOn)

  Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)

  Op(ACTION: LeftSock, EFFECT: leftSockOn)

could result in a partial plan of

[RightShoe, LeftShoe]

Partial-order planning

- A **linear planner** builds a plan as a **totally ordered sequence** of plan steps
- A **non-linear planner (aka partial-order planner)** builds up a plan as a set of steps with some temporal constraints
  - constraints of the form S1<S2 if step S1 must comes before S2.
- One **refines** a partially ordered plan (POP) by either:
  - **adding a new plan step**, or
  - **adding a new constraint** to the steps already in the plan.
- A POP can be **linearized** (converted to a totally ordered plan) by topological sorting

Least commitment

- Non-linear planners embody the principle of **least commitment**
  - only choose actions, orderings, and variable bindings that are absolutely necessary, leaving other decisions till later
  - avoids early commitment to decisions that don't really matter
- A linear planner always chooses to add a plan step in a particular place in the sequence
- A non-linear planner chooses to add a step and possibly some temporal constraints

Non-linear plan

- A non-linear plan consists of

  (1) A set of **steps** {S_1, S_2, S_3, S_4…}

  Each step has an operator description, preconditions and post-conditions

  (2) A set of **causal links** { … (S_i,C,S_j) …}

  Meaning a purpose of step S_i is to achieve precondition C of step S_j

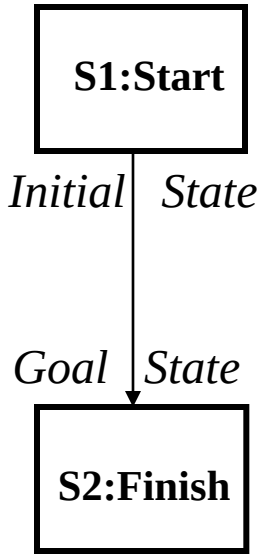
  (3) A set of **ordering constraints** { … S_i<S_j … }

  if step S_i must come before step S_j

- A non-linear plan is **complete** iff
  - Every step mentioned in (2) and (3) is in (1)
  - If S_j has prerequisite C, then there exists a causal link in (2) of the form (S_i,C,S_j) for some S_i
  - If (S_i,C,S_j) is in (2) and step S_k is in (1), and S_k threatens (S_i,C,S_j) (makes C false), then (3) contains either S_k<S_i or S_j>S_k

The initial plan

Every plan starts the same way



Trivial example

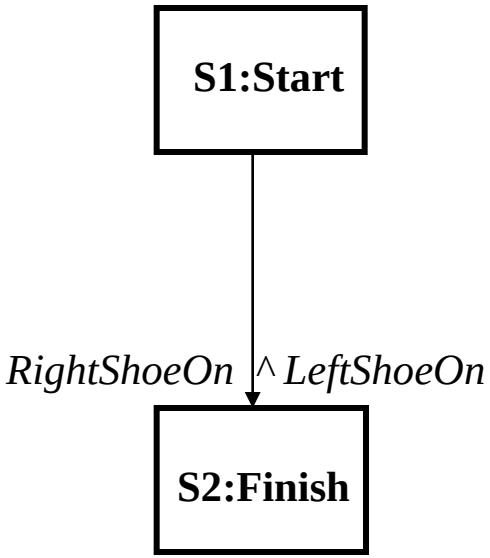
Operators:

Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)

Op(ACTION: RightSock, EFFECT: RightSockOn)

Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)

Op(ACTION: LeftSock, EFFECT: leftSockOn)



Steps: {S1:[Op(Action:Start)],

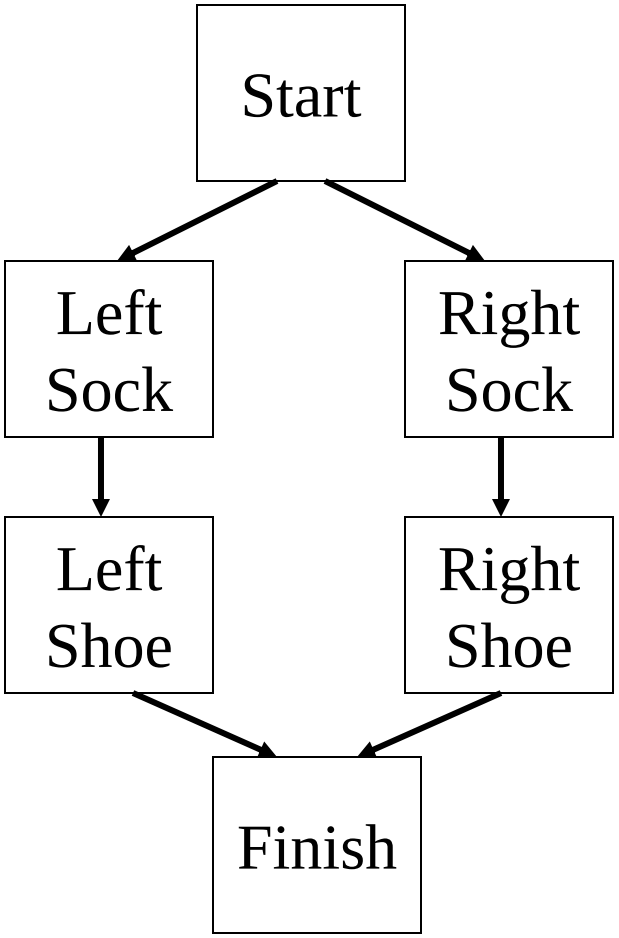
S2:[Op(Action:Finish,

Pre: RightShoeOn^LeftShoeOn)]}

Links: {}

Orderings: {S1<S2}

Solution

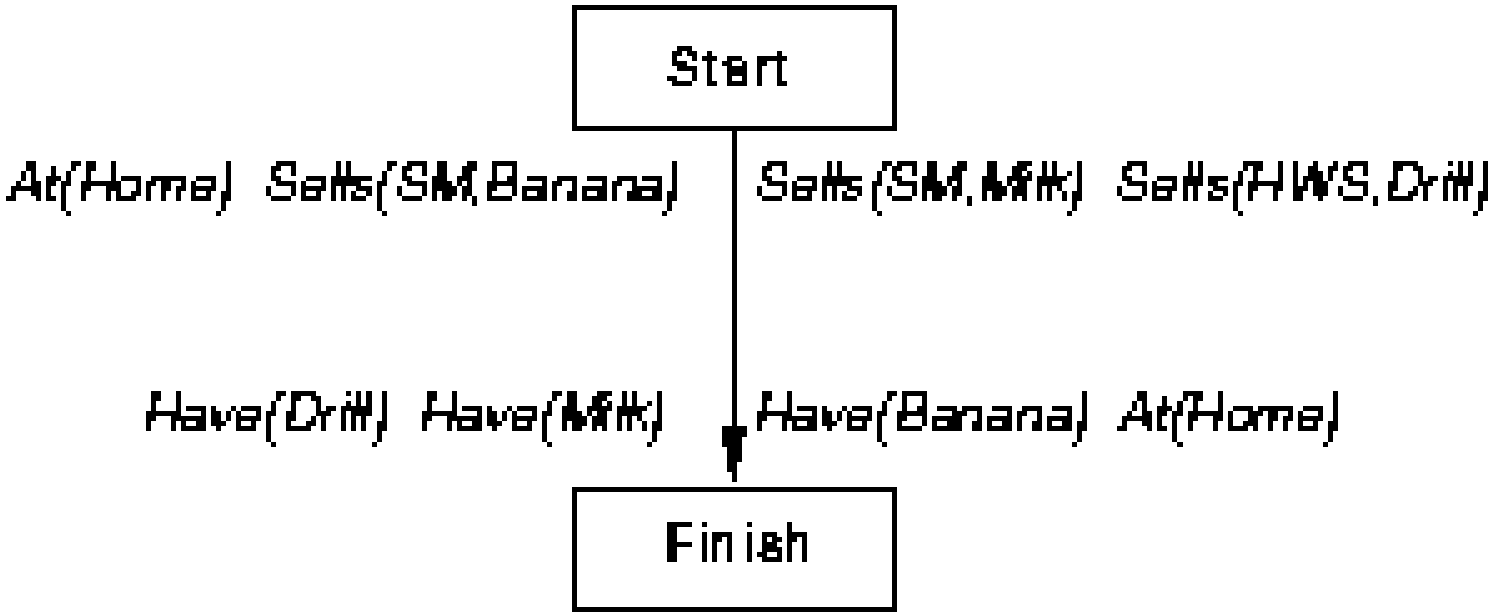


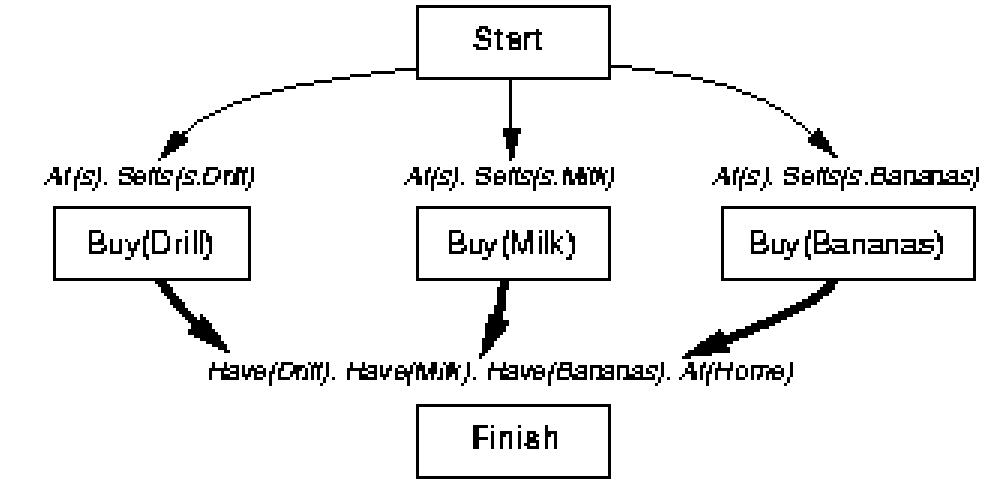
POP constraints and search heuristics

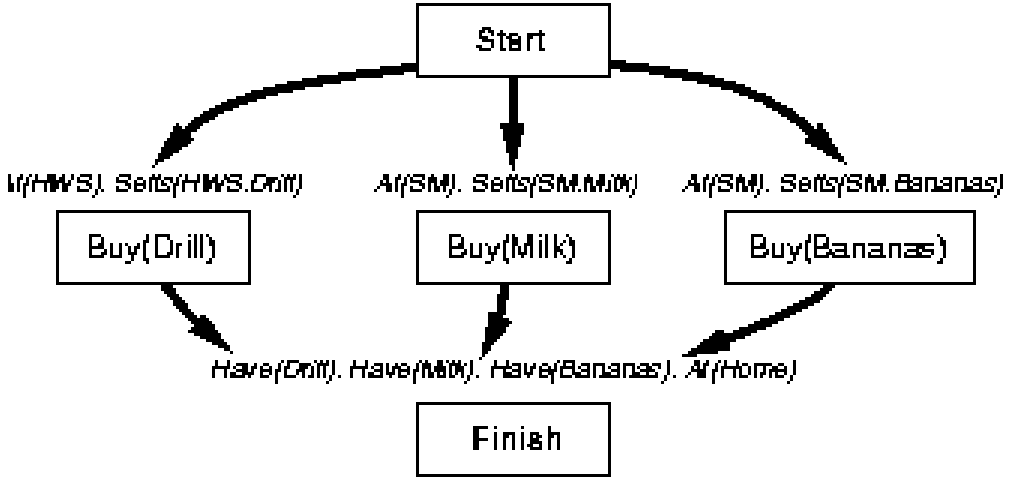
- Only add steps that achieve a currently unachieved precondition
- Use a least-commitment approach:
  - Don't order steps unless they need to be ordered
- Honor causal links $S_1 \xrightarrow{c} S_2$ that **protect** a condition *c*:
  - Never add an intervening step S_3 that violates *c*
  - If a parallel action **threatens** *c* (i.e., has the effect of negating or **clobbering** *c*), resolve that threat by adding ordering links:
    - Order S_3 before S_1 (**demotion**)
    - Order S_3 after S_2 (**promotion**)

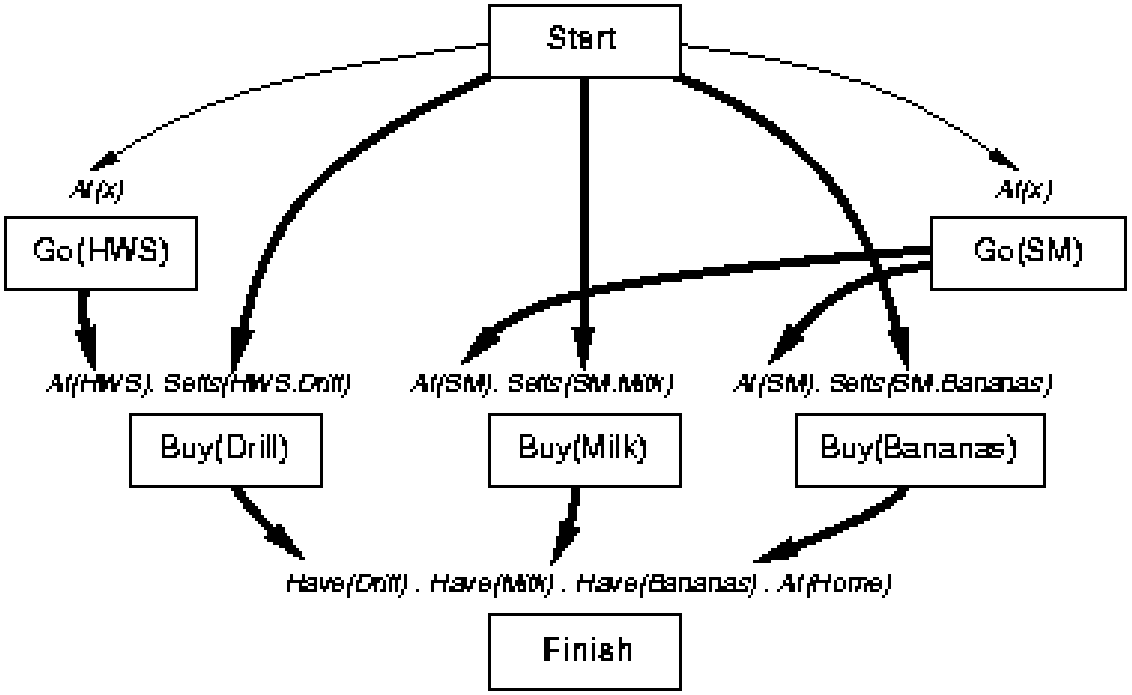
Partial-order planning example

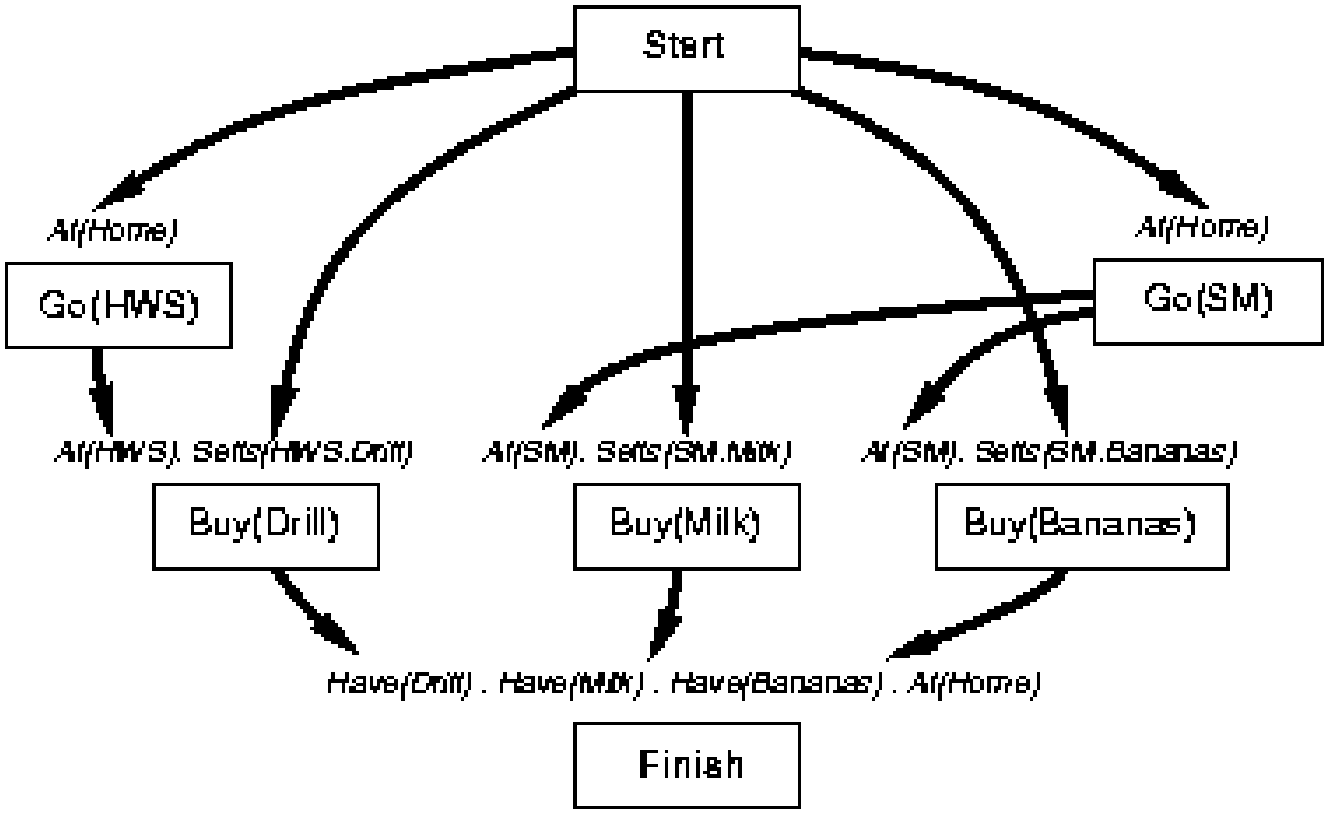
- Goal: Have milk, bananas, and a drill


Start
At(Home)
Sells(SM,Banana)
Sells(SM,Milk)
Sells(HWS,Drill)
Have(Drill)
Have(Milk)
Have(Banana)
At(Home)
Finish

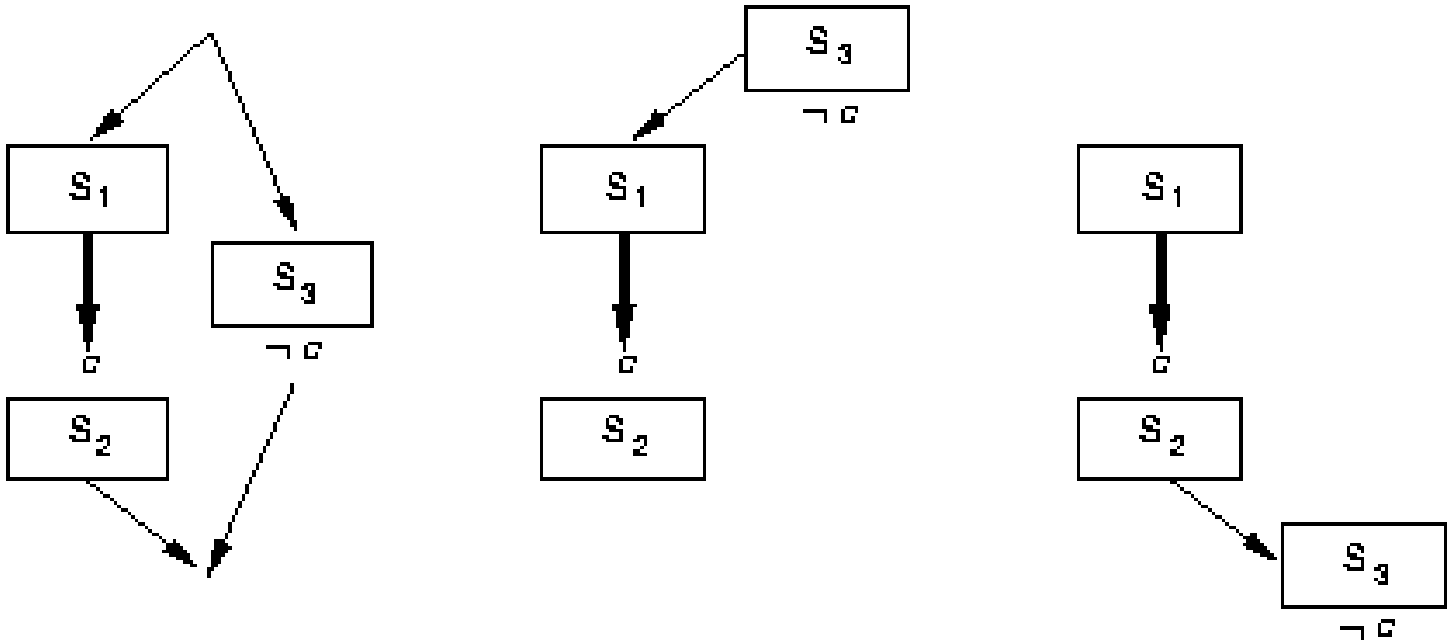

Start
At(s). Sells(s,Drill)
At(s). Sells(s,Milk)
At(s). Sells(s,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill). Have(Milk). Have(Bananas). At(Home)
Finish


Start
At(HWS). Sells(HWS,Drill)
At(SM). Sells(SM,Milk)
At(SM). Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill). Have(Milk). Have(Bananas). At(Home)
Finish

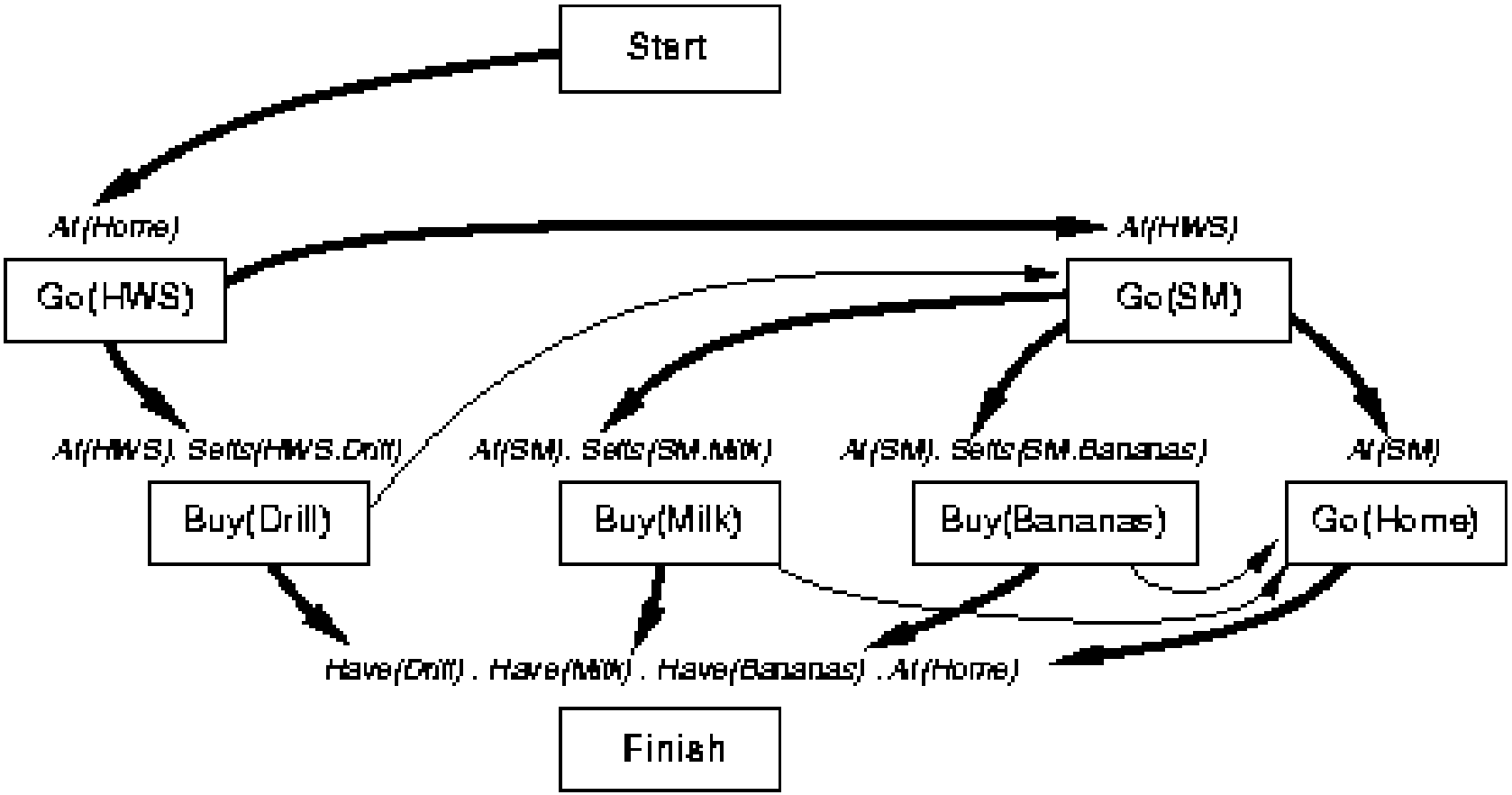

Start
At(x)
At(x)
Go(HWS)
Go(SM)
At(HWS). Sells(HWS,Drill)
At(SM). Sells(SM,Milk)
At(SM). Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) . Have(Milk) . Have(Bananas) . At(Home)
Finish

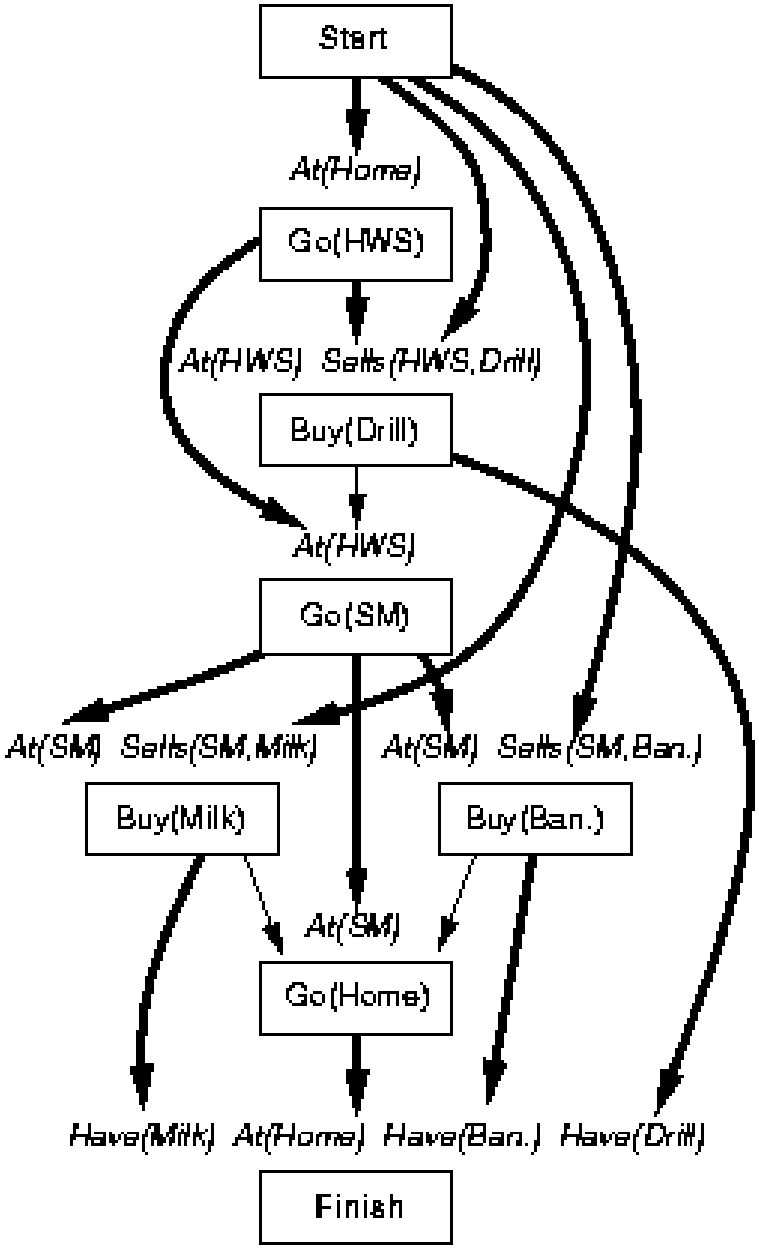

Start
At(Home)
Go(HWS)
At(Home)
Go(SM)
At(HWS). Sells(HWS,Drill)
At(SM). Sells(SM,Milk)
At(SM). Sells(SM,Bananas)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Have(Drill) . Have(Milk) . Have(Bananas) . At(Home)
Finish

Resolving threats



(a) Threat (b) Demotion (c) Promotion


Start
At(Home)
At(HWS)
Go(HWS)
Go(SM)
At(HWS). Sells(HWS.Drill)
At(SM). Sells(SM.Milk)
At(SM). Sells(SM.Bananas)
At(SM)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Go(Home)
Have(Drill) . Have(Milk) . Have(Bananas) . At(Home)
Finish

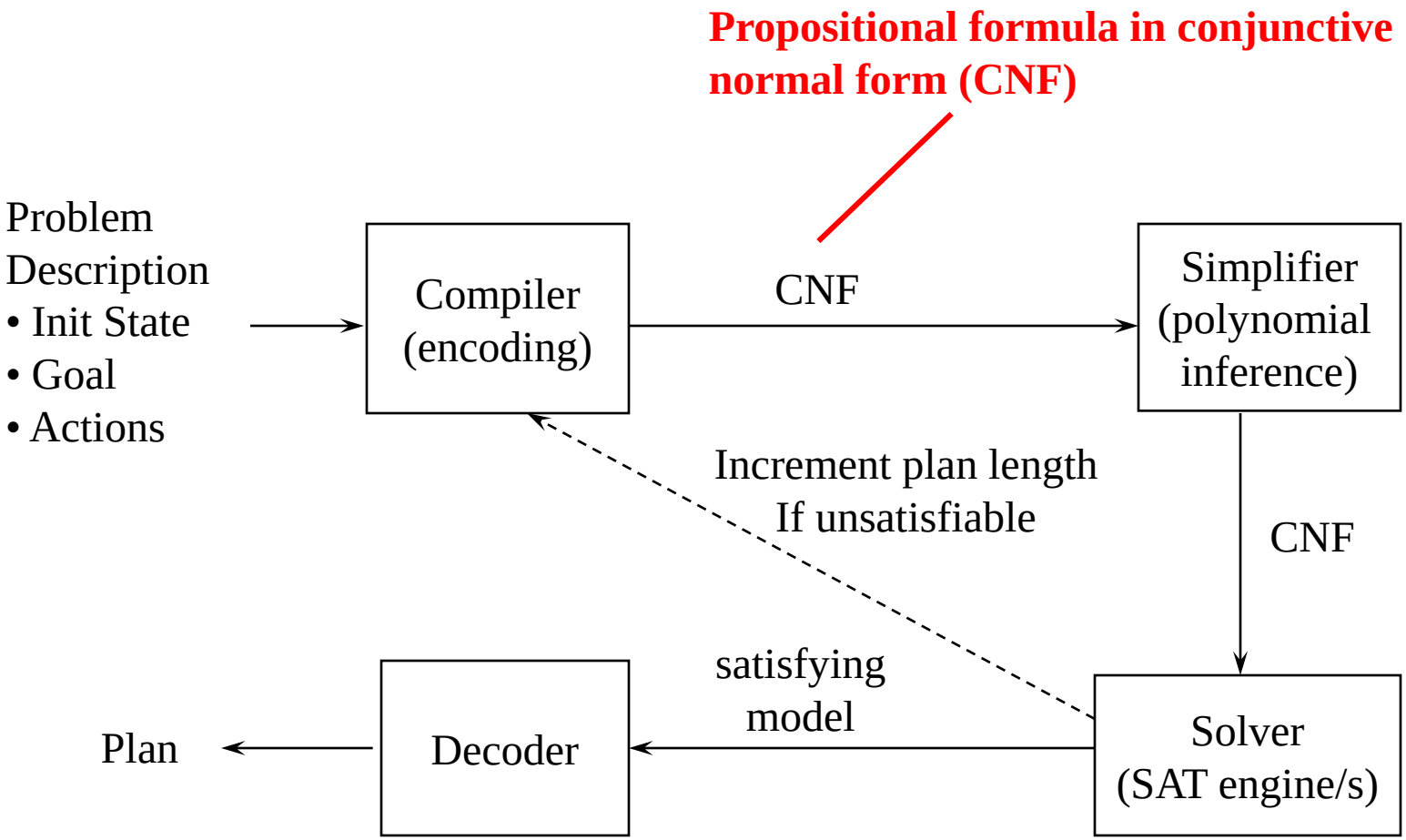

Start
At(Home)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(HWS)
Go(SM)
At(SM) Sells(SM,Milk)
At(SM) Sells(SM,Ban.)
Buy(Milk)
Buy(Ban.)
At(SM)
Go(Home)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish

Planning as Satisfiability

- Planning as propositional satisfiability

* Based on slides by Alan Fern, Stuart Russell and Dana Nau

Architecture of a SAT-based Planner



Propositional Satisfiability

- A formula is ***satisfiable*** if it is true in some model
  - e.g. $A \vee B, C$
- A formula is unsatisfiable if it is true in no models
  - e.g. $A \wedge \neg A$
- Testing satisfiability of CNF formulas is a famous NP-complete problem

Propositional Satisfiability

- Many problems (such as planning) can be naturally encoded as instances of satisfiability
- Thus there has been much work on developing powerful satisfiability solvers
  - these solvers work amazingly well in practice (we will touch on some later)

Encoding Planning as Satisfiability: Basic Idea

- Bounded planning problem (P,n):
  - P is a planning problem; n is a positive integer
  - Find a solution for P of length n
- Create a propositional formula that represents:
  - Initial state
  - Goal
  - Action Dynamics

  for n time steps
- We will define the formula for (P,n) such that:
  1) ***any*** model (i.e. satisfying truth assignment) of the formula represent a solution to (P,n)
  2) if (P,n) has a solution then the formula is satisfiable

Encoding Planning Problems

- We can encode (P,n) so that we consider either partially or totally ordered plans
  - for simplicity we consider totally-ordered plans
- Encode (P,n) as a formula Φ such that

  $\langle a_0, a_1, \ldots, a_{n-1} \rangle$ 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
- Φ will be conjunction of many other formulas …

Formulas in Φ

- Formula describing the **initial state**: (let E be the set of possible facts in the planning problem)

  $\Lambda\{e_0 \mid e \in s_0\} \wedge \Lambda\{\neg e_0 \mid e \in E - s_0\}$

  Describes the complete initial state (both positive and negative fact)

  – E.g. $on(A,B,0) \wedge \neg on(B,A,0)$

- Formula describing the **goal**: (G is set of goal facts)

  $\Lambda\{e_n \mid e \in G\}$

  says that the goal facts must be true in the final state at timestep n

  – E.g. $on(B,A,n)$

- Is this enough?

  – Of course not. The formulas say nothing about actions.

Formulas in Φ

- For every*a* and timestep *i*, formula describing what fluents must be true if *a* action were the *i*'th step of the plan:
  - $a_i \Rightarrow \bigwedge \{e_i \mid e \in \text{Precond}(a)\}$, *a*'s preconditions must be true
  - $a_i \Rightarrow \bigwedge \{e_{i+1} \mid e \in \text{ADD(a)}\}$, a's ADD effects must be true in i+1
  - $a_i \Rightarrow \bigwedge \{\neg e_{i+1} \mid e \in \text{DEL}(a)\}$, a's DEL effects must be false in i+1
- *Complete exclusion* axiom:
  - For all actions *a* and *b* and timesteps *i*, formulas saying *a* and *b* 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?
  - The formulas say nothing about what happens to facts if they are not effected by an action
  - This is the ***frame problem*** again**.**

Frame Axioms

- *Frame axioms*:
  - Formulas describing what *doesn't* change between steps *i* and *i*+1
- Several ways to write these (your book shows another way)
  - Here I show a alternative that typically works best in practice
- **explanatory frame axioms**
  - One axiom for every possible fact *e at every* timestep *i*
  - Says that if *e* changes truth value between s_i and s_{i+1}, then the action at step *i* must be responsible:

$$\neg e_i \wedge e_{i+1} \Rightarrow \mathrm{V}\{a_i \mid \mathrm{e}\ \mathit{in}\ \mathrm{ADD}(a)\}$$

If *e* became true then some action must have added it

$$e_i \wedge \neg e_{i+1} \Rightarrow \mathrm{V}\{a_i \mid \mathrm{e}\ \mathit{in}\ \mathrm{DEL}(a)\}$$

If *e* became false then some action must have deleted it

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) $\wedge$ $\neg$at(r1,l2,0)

  - Goal: {at(r1,l2)}
    Encoding: at(r1,l2,1)

  - Action Schema: see next slide

Example (continued)

- Schema: move(r, l, l')

  PRE: at(r,l)

  ADD: at(r,l')

  DEL: at(r,l)

Encoding: (for actions move(r1,l1,l2) and move(r1,l2,l1) at time step 0)

$move(r1,l1,l2,0) \Rightarrow at(r1,l1,0)$

$move(r1,l1,l2,0) \Rightarrow at(r1,l2,1)$

$move(r1,l1,l2,0) \Rightarrow \neg at(r1,l1,1)$

$move(r1,l2,l1,0) \Rightarrow at(r1,l2,0)$

$move(r1,l2,l1,0) \Rightarrow at(r1,l1,1)$

$move(r1,l2,l1,0) \Rightarrow \neg at(r1,l2,1)$

Example (continued)

- Schema: move(r, l, l')
  PRE: at(r,l)
  ADD: at(r,l')
  DEL: at(r,l)

- Complete-exclusion axiom:
  $\neg$move(r1,l1,l2,0) $\vee$ $\neg$move(r1,l2,l1,0)

- Explanatory frame axioms:
  $\neg$at(r1,l1,0) $\wedge$ at(r1,l1,1) $\Rightarrow$ move(r1,l2,l1,0)
  $\neg$at(r1,l2,0) $\wedge$ at(r1,l2,1) $\Rightarrow$ move(r1,l1,l2,0)
  at(r1,l1,0) $\wedge$ $\neg$at(r1,l1,1) $\Rightarrow$ move(r1,l1,l2,0)
  at(r1,l2,0) $\wedge$ $\neg$at(r1,l2,1) $\Rightarrow$ move(r1,l2,l1,0)

Complete Formula for (*P*,1)

$[\ at(r1,l1,0) \wedge \neg at(r1,l2,0)\ ] \wedge$
$at(r1,l2,1) \wedge$
$[\ move(r1,l1,l2,0) \Rightarrow at(r1,l1,0)\ ] \wedge$
$[\ move(r1,l1,l2,0) \Rightarrow at(r1,l2,1)\ ] \wedge$
$[\ move(r1,l1,l2,0) \Rightarrow \neg at(r1,l1,1)\ ] \wedge$
$[\ move(r1,l2,l1,0) \Rightarrow at(r1,l2,0)\ ] \wedge$
$[\ move(r1,l2,l1,0) \Rightarrow at(r1,l1,1)\ ] \wedge$
$[\ move(r1,l2,l1,0) \Rightarrow \neg at(r1,l2,1)\ ] \wedge$
$[\ \neg move(r1,l1,l2,0) \vee \neg move(r1,l2,l1,0)\ ] \wedge$
$[\ \neg at(r1,l1,0) \wedge at(r1,l1,1) \Rightarrow move(r1,l2,l1,0)\ ] \wedge$
$[\ \neg at(r1,l2,0) \wedge at(r1,l2,1) \Rightarrow move(r1,l1,l2,0)\ ] \wedge$
$[\ at(r1,l1,0) \wedge \neg at(r1,l1,1) \Rightarrow move(r1,l1,l2,0)\ ] \wedge$
$[\ at(r1,l2,0) \wedge \neg at(r1,l2,1) \Rightarrow move(r1,l2,l1,0)\ ]$

Convert to CNF and give to SAT solver.

Extracting a Plan

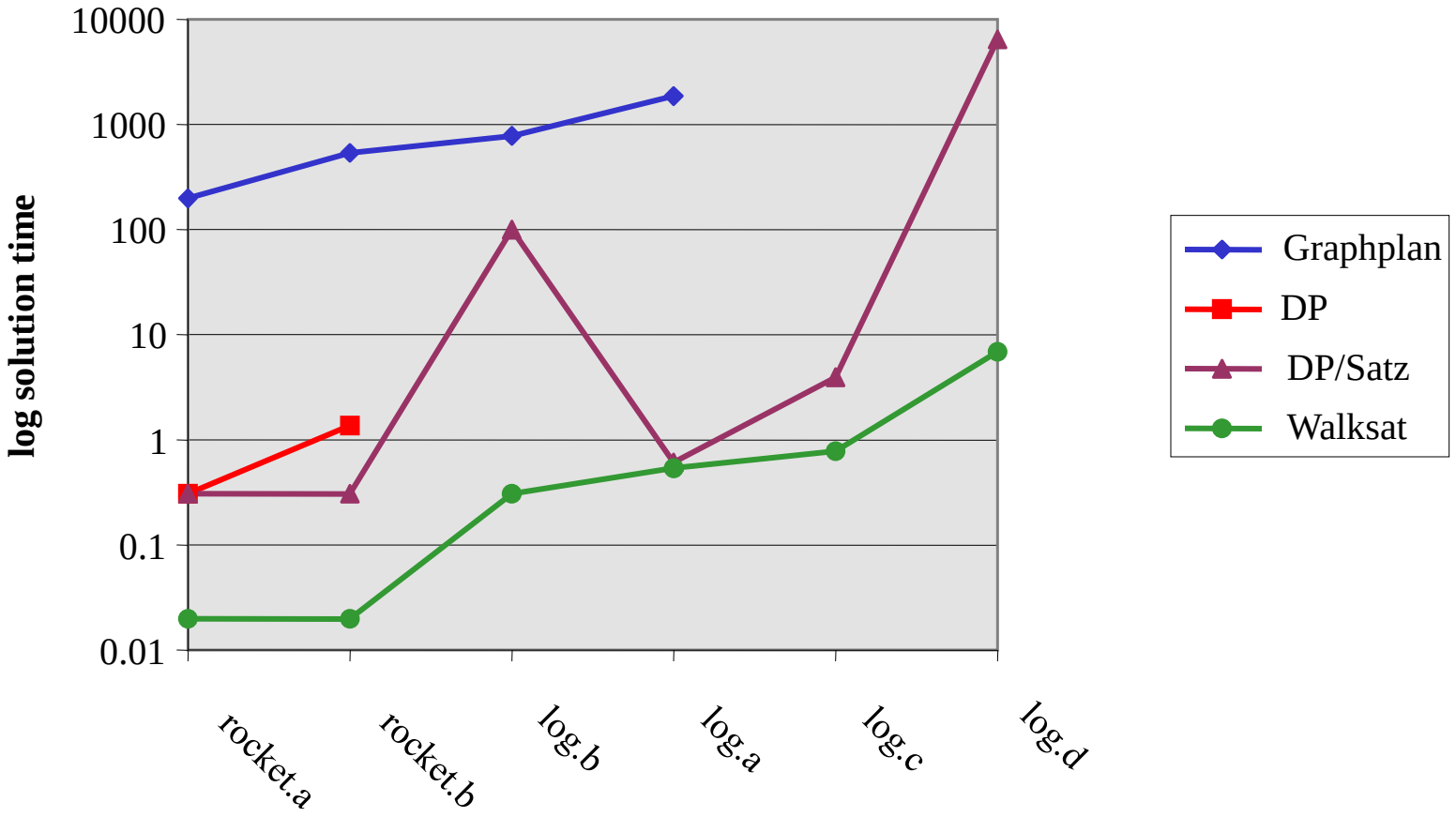
- Suppose we find an assignment of truth values that satisfies Φ.
  - This means *P* has a solution of length *n*
- For *i*=0,…,*n-1*, there will be exactly one action *a* such that a_i = *true*
  - This is the *i*'th action of the plan.
- Example (from the previous slides):
  - Φ can be satisfied with 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 Benchmark Test Set

- Extension of Graphplan test set
- blocks world - up to 18 blocks, 10^{19} states
- logistics - complex, highly-parallel transportation domain.

  Logistics.d:
  - 2,165 possible actions per time slot
  - 10^{16} legal configurations (2^{2000} states)
  - optimal solution contains 74 distinct actions over 14 time slots
- *Problems of this size never previously handled by general-purpose planning systems*

Scaling Up Logistics Planning



What SATPLAN Shows

- General propositional reasoning can compete with state of the art specialized planning systems
  - New, highly tuned variations of DP surprising powerful
  - Radically new stochastic approaches to SAT can provide very low exponential scaling
- Why does it work?
  - More flexible than forward or backward chaining
  - Randomized algorithms less likely to get trapped along bad paths

Discussion

- How well does this work?
  - Created an initial splash but by itself, not very practical without help in choosing good encoding
- However combining SatPlan with planning graphs can overcome this problem