CS343
Artificial Intelligence

Prof: Peter Stone

Department of Computer Science
The University of Texas at Austin

Good Morning, Colleagues

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  – Find an optimal plan (or solution)
  – Best thing to do from the current state
  – Know transition and cost (reward) functions
  – Either execute complete solution (deterministic) or search again at every step

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**
- **Next:** MDPs —

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**
- **Next:** MDPs — towards reinforcement learning

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**
- **Next:** MDPs — towards reinforcement learning
  - Still know transition and reward function

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**
- **Next:** MDPs — towards reinforcement learning
  - Still know transition and reward function
  - Looking for a **policy** — optimal action from every state

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**
- **Next:** MDPs — towards reinforcement learning
  - Still know transition and reward function
  - Looking for a **policy** — optimal action from every state
- **Action learning:** Reinforcement learning

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**
- **Next:** MDPs — towards reinforcement learning
  - Still know transition and reward function
  - Looking for a **policy** — optimal action from every state
- **Action learning:** Reinforcement learning
  - Policy without knowing transition or reward functions

Some Context

- **First weeks:** search (BFS, A*, minimax, alpha-beta)
  - Find an optimal plan (or solution)
  - Best thing to do from the current state
  - Know transition and cost (reward) functions
  - Either execute complete solution (deterministic) or search again at every step
  - **Know current state**
- **Next:** MDPs — towards reinforcement learning
  - Still know transition and reward function
  - Looking for a **policy** — optimal action from every state
- **Action learning:** Reinforcement learning
  - Policy without knowing transition or reward functions
  - **Still know state**

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**
  - **Week 4:** Utilities

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**
  - **Week 4:** Utilities
  - **Week 7:** Conditional independence and inference (exact and approximate)

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**
  - **Week 4:** Utilities
  - **Week 7:** Conditional independence and inference (exact and approximate)
  - **Week 9:** State estimation over time

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**
  - **Week 4:** Utilities
  - **Week 7:** Conditional independence and inference (exact and approximate)
  - **Week 9:** State estimation over time
  - **Week 9:** Utility-based decisions

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**
  - **Week 4:** Utilities
  - **Week 7:** Conditional independence and inference (exact and approximate)
  - **Week 9:** State estimation over time
  - **Week 9:** Utility-based decisions
- **Week 10:** What if they're not known?

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**
  - **Week 4:** Utilities
  - **Week 7:** Conditional independence and inference (exact and approximate)
  - **Week 9:** State estimation over time
  - **Week 9:** Utility-based decisions
- **Week 10:** What if they're not known?
  - Also Bayesian networks for **classification**

Some Context (cont.)

- **Probabilistic Reasoning:** Now state is unknown
- Bayesian networks – state estimation/inference
- **Prior, net structure, and CPT's known**
  - **Week 4:** Utilities
  - **Week 7:** Conditional independence and inference (exact and approximate)
  - **Week 9:** State estimation over time
  - **Week 9:** Utility-based decisions
- **Week 10:** What if they're not known?
  - Also Bayesian networks for **classification**
  - A type of **machine learning**

Some Context (cont.)

- **After that:** More machine learning
  - **Week 11:** Neural nets and Deep Learning
  - **Week 12:** SVMs, Kernels, and Clustering
- **Week 13:** Classical planning
  - Reasoning with first order representations

Some Context (cont.)

- **After that:** More machine learning
  - **Week 11:** Neural nets and Deep Learning
  - **Week 12:** SVMs, Kernels, and Clustering
- **Week 13:** Classical planning
  - Reasoning with first order representations
  - So far we've dealt with propositions

Some Context (cont.)

- **After that:** More machine learning
  - **Week 11:** Neural nets and Deep Learning
  - **Week 12:** SVMs, Kernels, and Clustering
- **Week 13:** Classical planning
  - Reasoning with first order representations
  - So far we've dealt with propositions
  - Back to known transitions, known state, etc.

Some Context (cont.)

- **After that:** More machine learning
  - **Week 11:** Neural nets and Deep Learning
  - **Week 12:** SVMs, Kernels, and Clustering
- **Week 13:** Classical planning
  - Reasoning with first order representations
  - So far we've dealt with propositions
  - Back to known transitions, known state, etc.
- **Week 14:** Philosophical foundations and ethics

Some Context (cont.)

- **After that:** More machine learning
  - **Week 11:** Neural nets and Deep Learning
  - **Week 12:** SVMs, Kernels, and Clustering
- **Week 13:** Classical planning
  - Reasoning with first order representations
  - So far we've dealt with propositions
  - Back to known transitions, known state, etc.
- **Week 14:** Philosophical foundations and ethics

It's all about building agents

Sense, decide, act

Some Context (cont.)

- **After that:** More machine learning
  - **Week 11:** Neural nets and Deep Learning
  - **Week 12:** SVMs, Kernels, and Clustering
- **Week 13:** Classical planning
  - Reasoning with first order representations
  - So far we've dealt with propositions
  - Back to known transitions, known state, etc.
- **Week 14:** Philosophical foundations and ethics

It's all about building agents

Sense, decide, act **Maximize expected utility**

Topics not covered

- Knowledge representation and reasoning
  . (Chapters 7-9, 11, 12)
- Game theory and auctions (Sections 17.5, 17.6)
- Aspects of learning (Chapters 18, 19)
- Natural language (Chapters 22, 23)
- Vision (Chapter 24)
- Robotics (Chapter 25)

Planning

- Back to deciding what to **do** (actions)

Planning

- Back to deciding what to **do** (actions)
- Back to deterministic and static

Planning

- Back to deciding what to **do** (actions)
- Back to deterministic and static
  - But scales up

Planning

- Back to deciding what to **do** (actions)

- Back to deterministic and static
  - But scales up
  - It's used (don't be fooled by just 1 week in class)

Planning Applications

- Mobile robots
  - An initial motivator, and still being developed

Planning Applications

- Mobile robots
  - An initial motivator, and still being developed
- Simulated environments
  - Goal-directed agents for training or games

Planning Applications

- Mobile robots
  - An initial motivator, and still being developed
- Simulated environments
  - Goal-directed agents for training or games
- Web and grid environments
  - Composing queries or services
  - Workflows on a computational grid

Planning Applications

- Mobile robots
  - An initial motivator, and still being developed
- Simulated environments
  - Goal-directed agents for training or games
- Web and grid environments
  - Composing queries or services
  - Workflows on a computational grid
- Managing crisis situations
  - Oil-spill, forest fires, urban evacuation, in factories

Planning Applications

- Mobile robots
  - An initial motivator, and still being developed
- Simulated environments
  - Goal-directed agents for training or games
- Web and grid environments
  - Composing queries or services
  - Workflows on a computational grid
- Managing crisis situations
  - Oil-spill, forest fires, urban evacuation, in factories
- And many more
  - Factory automation, flying autonomous spacecraft, playing bridge, military planning,. . .

Questions

- What are some real-world applications?

Questions

- What are some real-world applications?
- Why not typically covered in intro AI?

Questions

- What are some real-world applications?
- Why not typically covered in intro AI?
- What's the relationship with...
  - CSPs?

Questions

- What are some real-world applications?
- Why not typically covered in intro AI?
- What's the relationship with...
  - CSPs?
  - heuristic search?

Questions

- What are some real-world applications?
- Why not typically covered in intro AI?
- What's the relationship with...
  - CSPs?
  - heuristic search?
  - reinforcement learning?

Questions

- What are some real-world applications?
- Why not typically covered in intro AI?
- What's the relationship with...
  - CSPs?
  - heuristic search?
  - reinforcement learning?
- Are there other types of planning?

Questions

- What are some real-world applications?
- Why not typically covered in intro AI?
- What's the relationship with...
  - CSPs?
  - heuristic search?
  - reinforcement learning?
- Are there other types of planning?
- What does it mean to be ground and functionless?

This Week

- Today: Planning problem representation
- Today: Solution types
- Today: Forward/backward search

This Week

- Today: Planning problem representation
- Today: Solution types
- Today: Forward/backward search
- Thursday: Heuristics
- Thursday: Graphplan

Good Morning, Colleagues

Questions

- Does classical planning have any applications for non-deterministic and continuous state/action spaces? (Conrad Li)

Questions

- Does classical planning have any applications for non-deterministic and continuous state/action spaces? (Conrad Li)

- Is it possible that along the path to a certain goal an action negates one of the literals in the eventual goal state (and then unnegates it in a later step)? And would this cause an issue for backward searching? (Lilia Li)

Questions

- What is a relaxed problem and how does it differ from a regular one? (Yash Kakodkar)
- Since some of the heuristics in 10.2 are not admissible, when would they ever be used? (Kelsey Zhan)

Questions

- How are heuristics from planning graphs always admissible? (Kelsey Zhan)
- How is the serial planning graph for heuristics better than using a normal planning graph?
- Why use planning graphs for heuristics when you can just extract a solution from them using the Graphplan algorithm? (Austin Aurelio)

Questions

- How are heuristics from planning graphs always admissible? (Kelsey Zhan)
- How is the serial planning graph for heuristics better than using a normal planning graph?
- Why use planning graphs for heuristics when you can just extract a solution from them using the Graphplan algorithm? (Austin Aurelio)

Planning

Chapter 11

Outline

♢ Search vs. planning

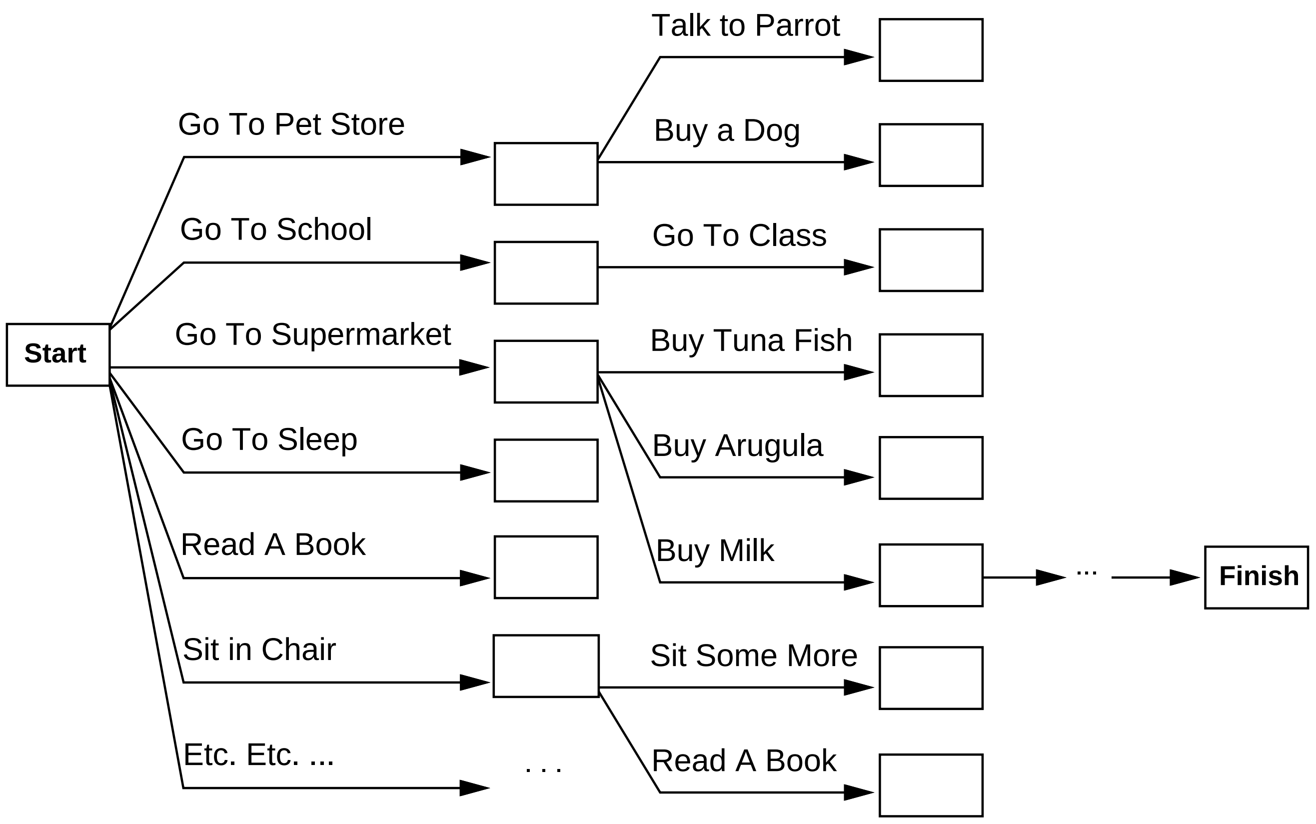
♢ STRIPS operators

♢ Partial-order planning

Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

Search vs. planning contd.

Planning systems do the following:

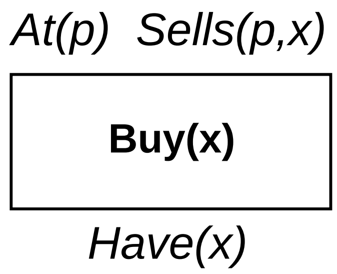
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

STRIPS operators

Tidily arranged actions descriptions, restricted language

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$



[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms

Partially ordered plans

Partially ordered collection of steps with

- $Start$ step has the initial state description as its effect
- $Finish$ step has the goal description as its precondition
- causal links from outcome of one step to precondition of another
- temporal ordering between pairs of steps

Open condition = precondition of a step not yet causally linked

A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step
and no possibly intervening step undoes it

Example

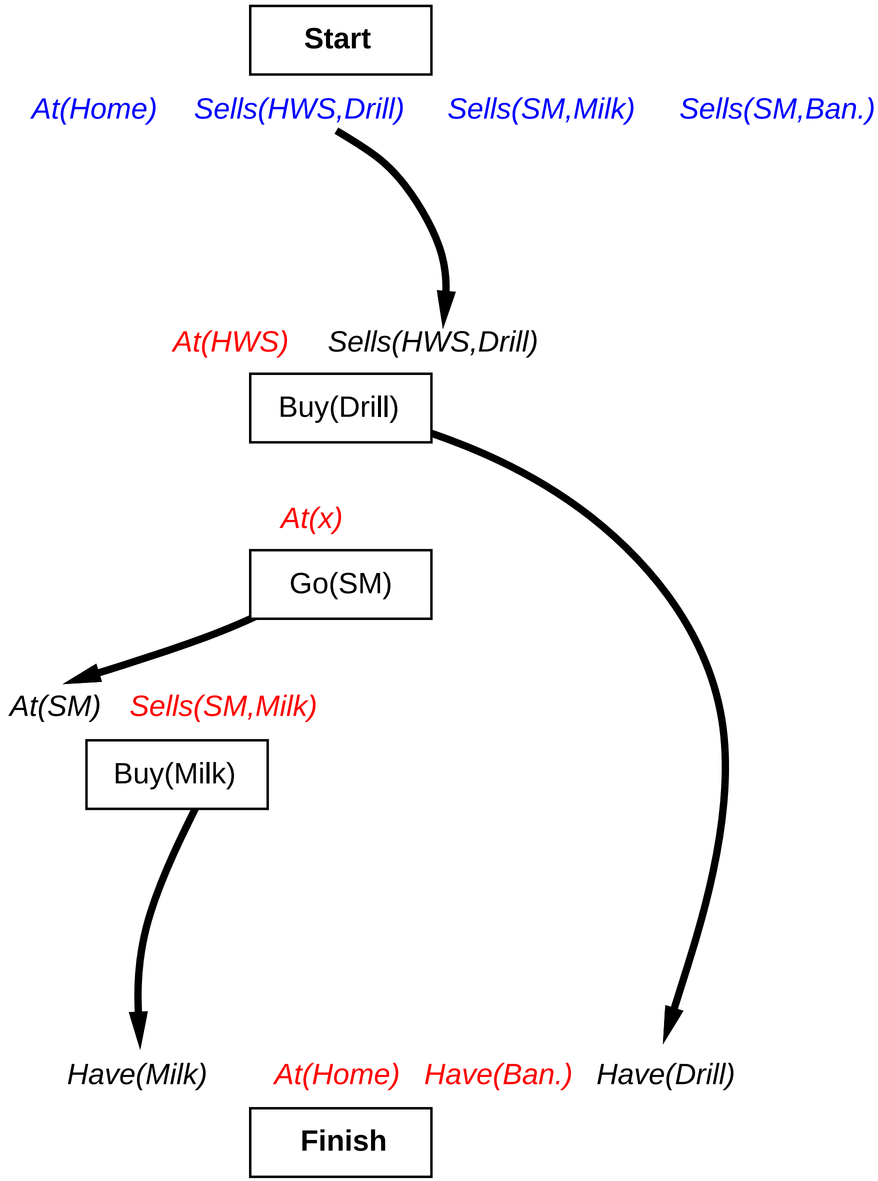
Start

At(Home) *Sells(HWS,Drill)* *Sells(SM,Milk)* *Sells(SM,Ban.)*

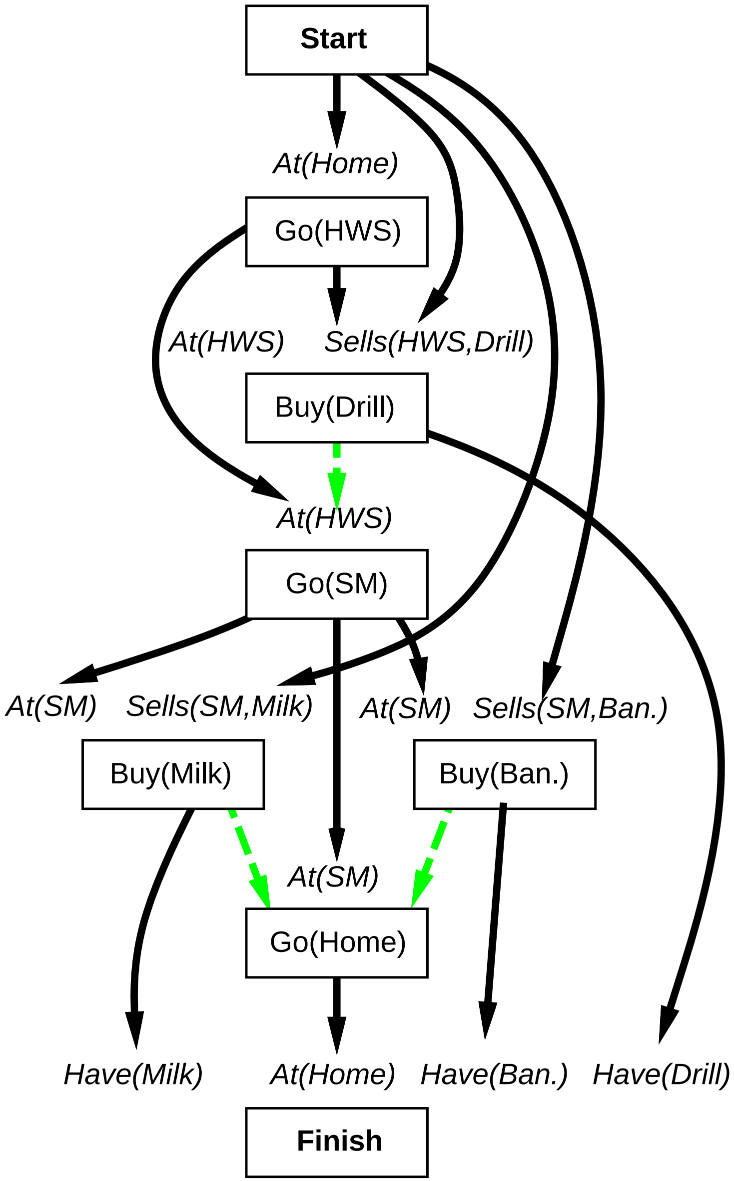
Have(Milk) *At(Home)* *Have(Ban.)* *Have(Drill)*

Finish

Example


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

Example


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 process

Operators on partial plans:

add a link from an existing action to an open condition
add a step to fulfill an open condition
order one step wrt another to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or
if a conflict is unresolvable

POP algorithm sketch

```
function POP(initial, goal, operators) returns plan

    plan ← Make-Minimal-Plan(initial, goal)
    loop do
        if Solution?( plan) then return plan
        S_need, c ← Select-Subgoal( plan)
        Choose-Operator( plan, operators, S_need, c)
        Resolve-Threats( plan)
    end
```

```
function Select-Subgoal( plan) returns S_need, c

    pick a plan step S_need from Steps( plan)
        with a precondition c that has not been achieved
    return S_need, c
```

POP algorithm contd.

procedure CHOOSE-OPERATOR(*plan, operators*, S_{need}, *c*)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
add S_{add} to STEPS(*plan*)
add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

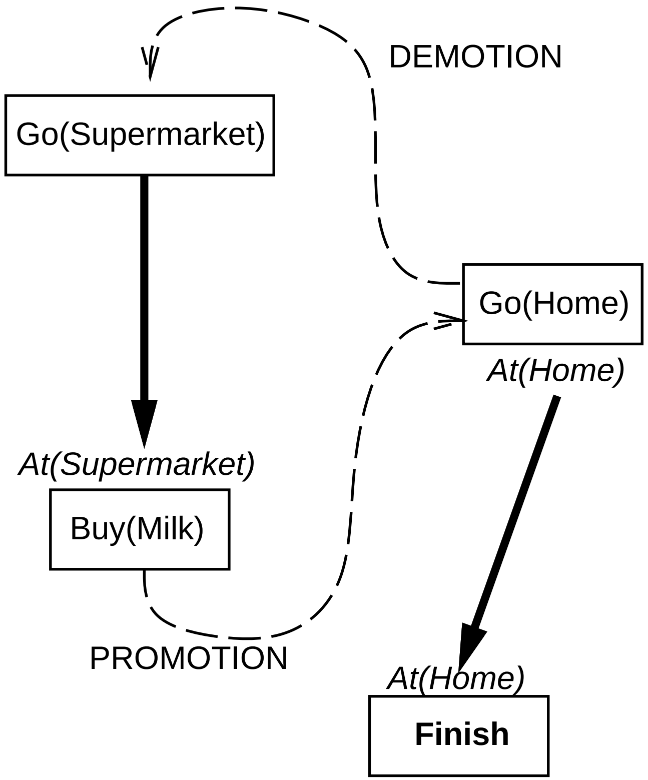
---

procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
choose either
Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
if not CONSISTENT(*plan*) **then fail**
end

Clobbering and promotion/demotion

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(Supermarket)$:



Demotion: put before $Go(Supermarket)$

Promotion: put after $Buy(Milk)$

Properties of POP

Nondeterministic algorithm: backtracks at choice points on failure:
- choice of S_{add} to achieve S_{need}
- choice of demotion or promotion for clobberer
- selection of S_{need} is irrevocable

POP is sound, complete, and systematic (no repetition)

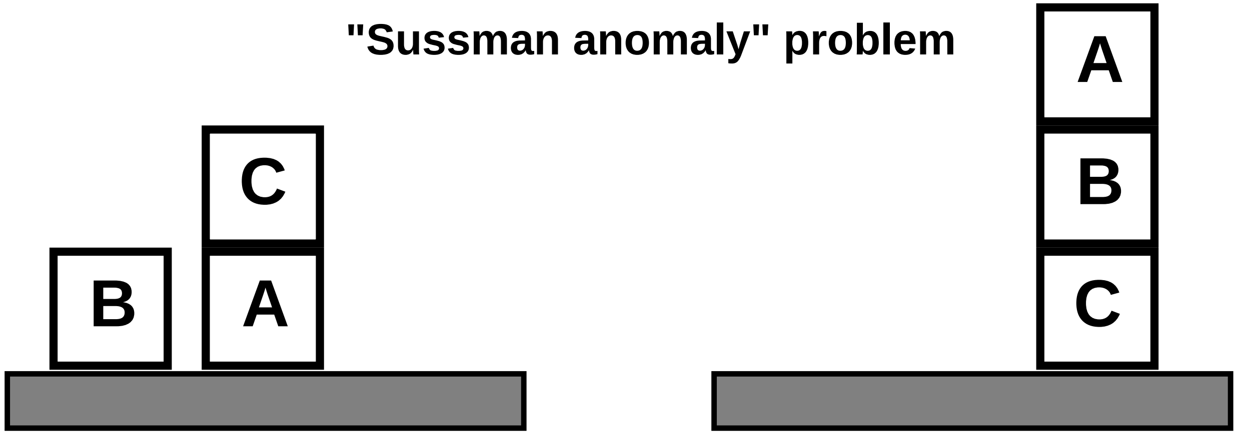
Extensions for disjunction, universals, negation, conditionals

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals

Example: Blocks world

"Sussman anomaly" problem



Start State Goal State

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

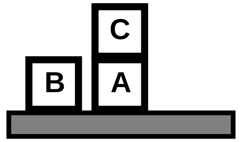
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Example contd.

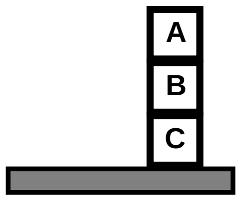
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

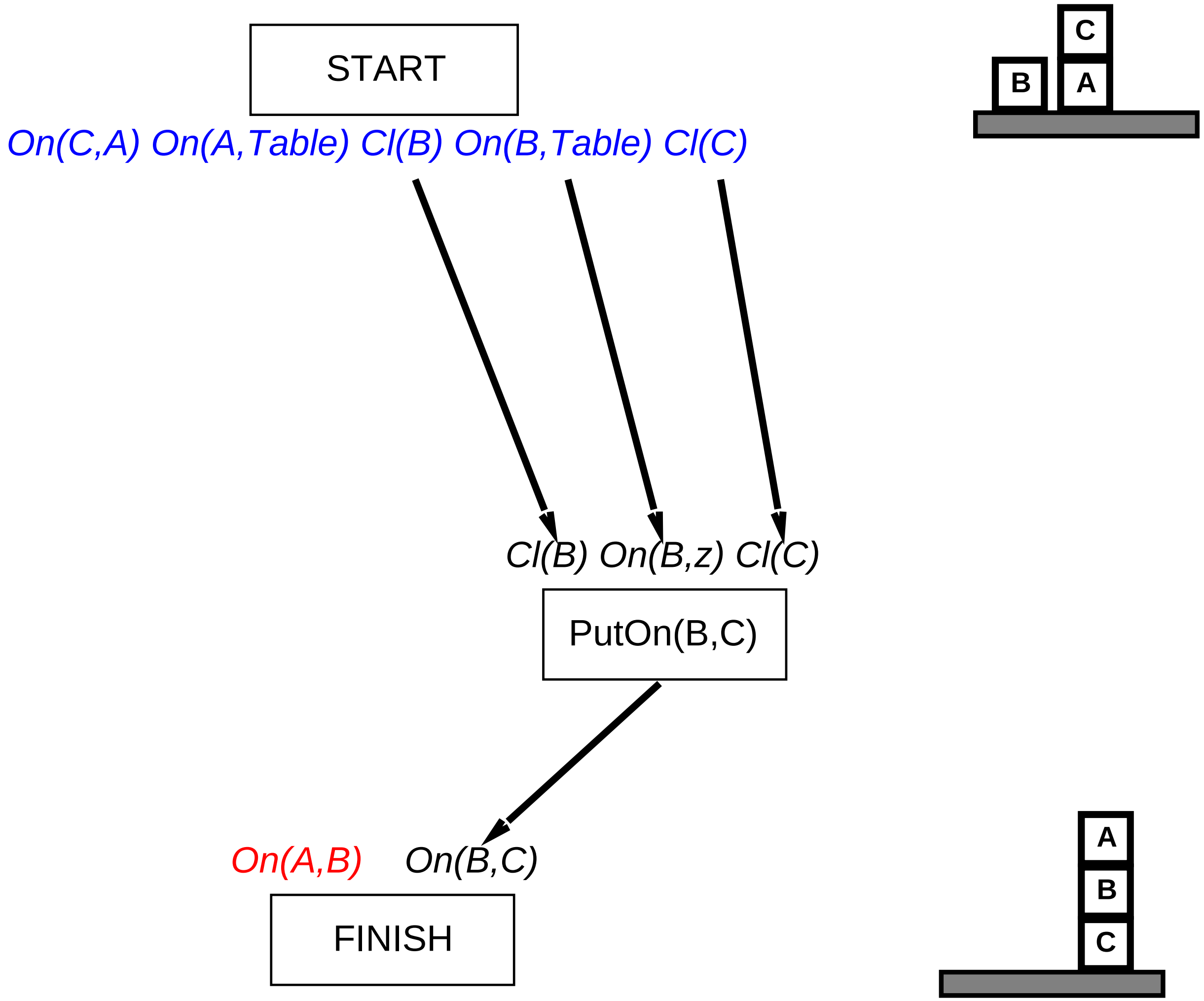


On(A,B)   *On(B,C)*

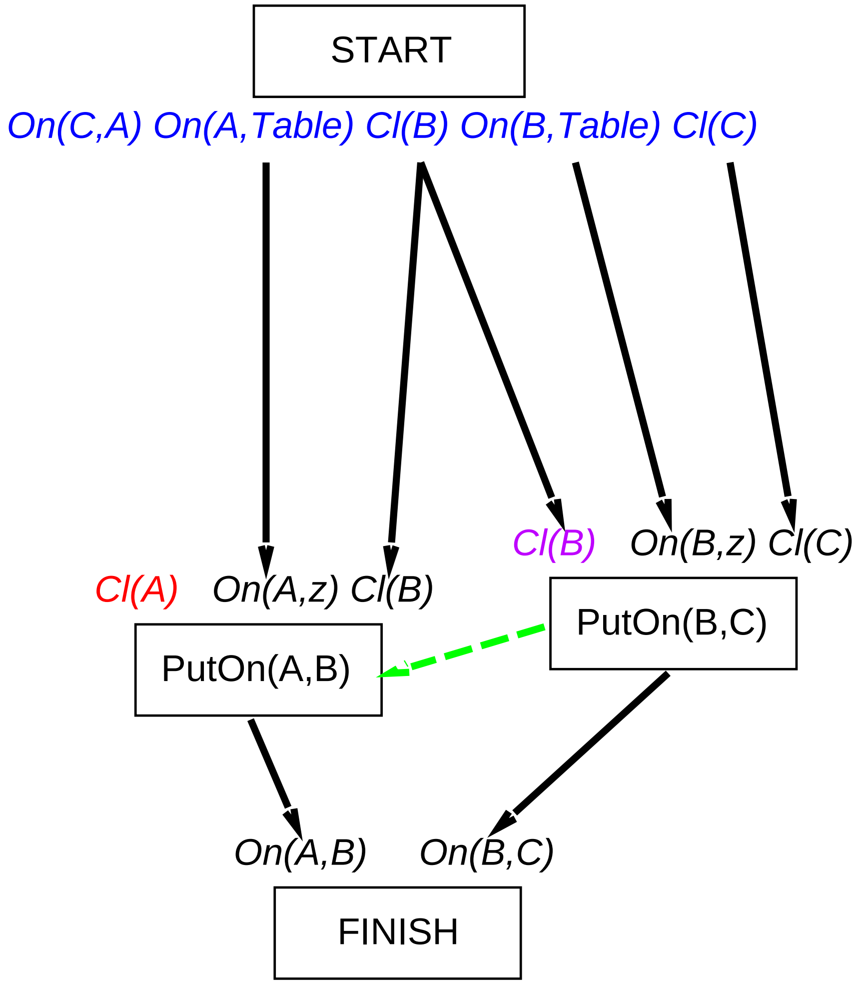
FINISH

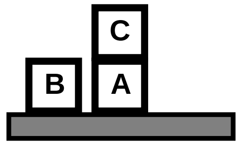


Example contd.

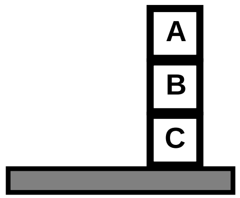


Example contd.

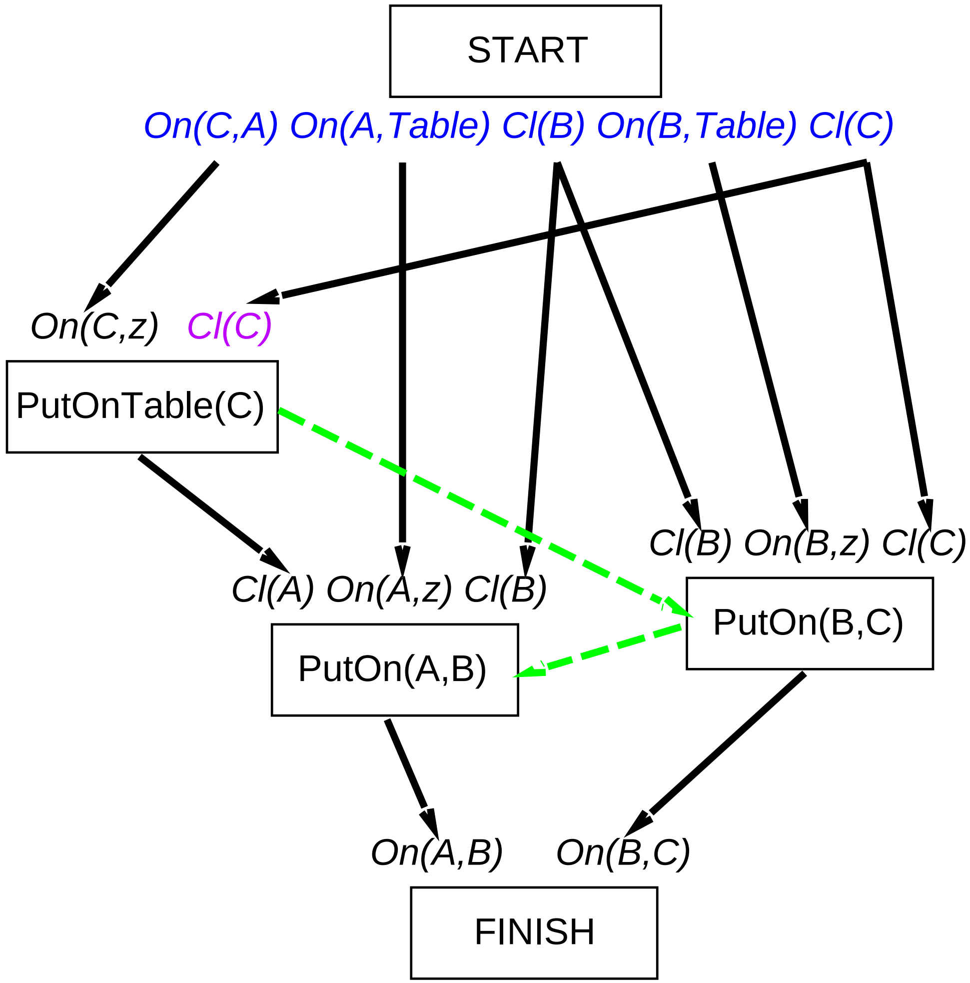


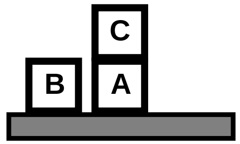


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



Example contd.





PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)