Planning

Chapter 11

First, some examples from Logic

Suppose a knowledge base contains just one sentence: $\exists x\, AsHighAs(x, Everest)$.

Which of the following are legitimate results of applying Existential Instantiation?

♢ $AsHighAs(Everest, Everest)$

♢ $AsHighAs(Kilimanjaro, Everest)$

♢ $AsHighAs(Kilimanjaro, Everest) \wedge AsHighAs(BenNevis, Everest)$
(after 2 applications)

First, some examples from Logic

Suppose a knowledge base contains just one sentence: $\exists x AsHighAs(x, Everest)$.

Which of the following are legitimate results of applying Existential Instantiation?

$\diamondsuit$ $AsHighAs(Everest, Everest)$

$\diamondsuit$ $AsHighAs(Kilimanjaro, Everest)$

$\diamondsuit$ $AsHighAs(Kilimanjaro, Everest) \wedge AsHighAs(BenNevis, Everest)$ (after 2 applications)

Numbers 2 and 3 are correct.
Why isn't number 1 correct?
Does number 3 mean there are now 2 mountains as high as $Everest$?

Another Logic example

"Brothers and sisters have I none, but that man's father is my father's son".
Who is that man?

Another Logic example

"Brothers and sisters have I none, but that man's father is my father's son".
Who is that man?

Let $Rel(r, x, y)$ say that family relationship r holds between x and y.

$Me =$ me, and $MrX =$ "that man".

(1) $Rel(Sibling, Me, x) \; \Rightarrow \; False$
(2) $Male(MrX)$
(3) $Rel(Father, FX, MrX)$
(4) $Rel(Father, FM, Me)$
(5) $Rel(Son, FX, FM)$

Another Logic example, con't.

Want to show that Me is the only son of my father, and therefore that Me is the father of MrX, who is male, and therefore that "that man" is my son.

(6) $Rel(Parent, x, y) \wedge Male(x) \iff Rel(Father, x, y)$
(7) $Rel(Son, x, y) \iff Rel(Parent, y, x) \wedge Male(x)$
(8) $Rel(Sibling, x, y) \iff x \neq y \wedge \exists p Rel(Parent, p, x) \wedge Rel(Parent, p, y)$
(9) $Rel(Father, x_1, y) \wedge Rel(Father, x_2, y) \Rightarrow x_1 = x_2$

Our query: (Q) $Rel(r, MrX, y)$

We want the answer: $\{r/Son, y/Me\}$

Another Logic example, con't.

Translating 1-9, we get:

(6a) $Rel(Parent, x, y) \land Male(x) \;\Rightarrow\; Rel(Father, x, y)$
(6b) $Rel(Father, x, y) \;\Rightarrow\; Male(x)$
(6c) $Rel(Father, x, y) \;\Rightarrow\; Rel(Parent, x, y)$
(7a) $Rel(Son, x, y) \;\Rightarrow\; Rel(Parent, y, x)$
(7b) $Rel(Son, x, y) \;\Rightarrow\; Male(x)$
(7c) $Rel(Parent, y, x) \land Male(x) \;\Rightarrow\; Rel(Son, x, y)$
(8a) $Rel(Sibling, x, y) \;\Rightarrow\; x \neq y$
(8b) $Rel(Sibling, x, y) \;\Rightarrow\; Rel(Parent, P(x, y), x)$
(8c) $Rel(Sibling, x, y) \;\Rightarrow\; Rel(Parent, P(x, y), y)$
(8d) $Rel(Parent, P(x, y), x) \land Rel(Parent, P(x, y), y) \land x \neq y \Rightarrow Rel(Sibling, x, y)$
(9) $Rel(Father, x_1, y) \land Rel(Father, x_2, y) \;\Rightarrow\; x_1 = x_2$
(N) $True \;\Rightarrow\; x = y \lor x \neq y$
(N') $x = y \land x \neq y \;\Rightarrow\; False$
(Q') $Rel(r, MrX, y) \;\Rightarrow\; False$

Another Logic example, con't.

Using resolution to prove Q' is a contradiction, we get the following:

(10: 4,6c) $Rel(Parent, FM, Me)$
(11: 5,7a) $Rel(Parent, FM, FX)$
(12: 10,8d) $Rel(Parent, FM, y) \wedge Me \neq y \ \Rightarrow \ Rel(Sibling, Me, y)$
(13: 12,1) $Rel(Parent, FM, y) \wedge Me \neq y \ \Rightarrow \ False$
(14: 13,11) $Me \neq FX \ \Rightarrow \ False$
(15: 14,N) $Me = FX$
(16: 15,3) $Rel(Father, Me, MrX)$
(17: 16,6c) $Rel(Parent, Me, MrX)$
(18: 17,2,7c) $Rel(Son, MrX, Me)$
(19: 18,Q') $False\{r/Son, y/Me\}$

Outline of Planning

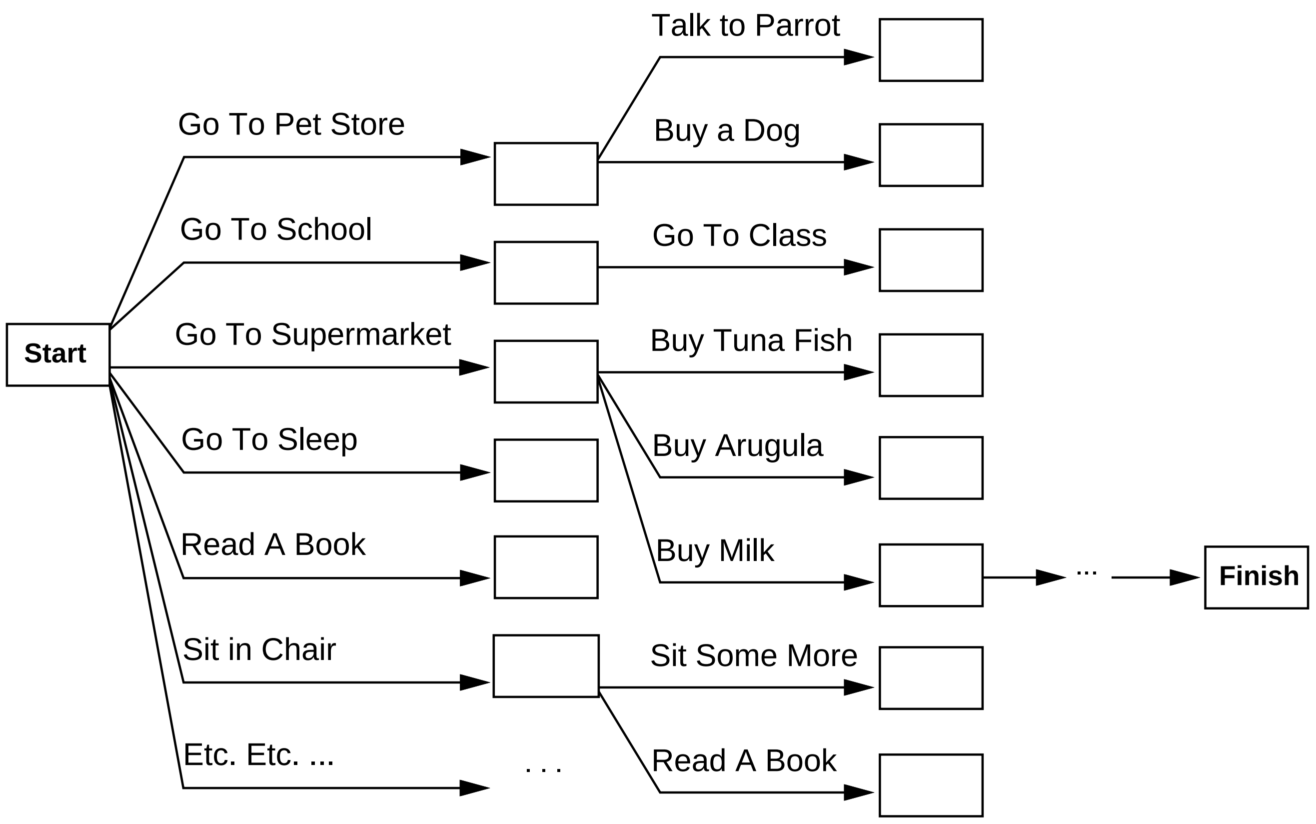
◊ Search vs. planning

◊ 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:

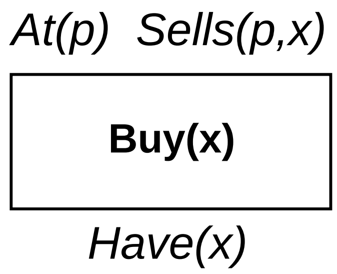
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

	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: $Have(x)$

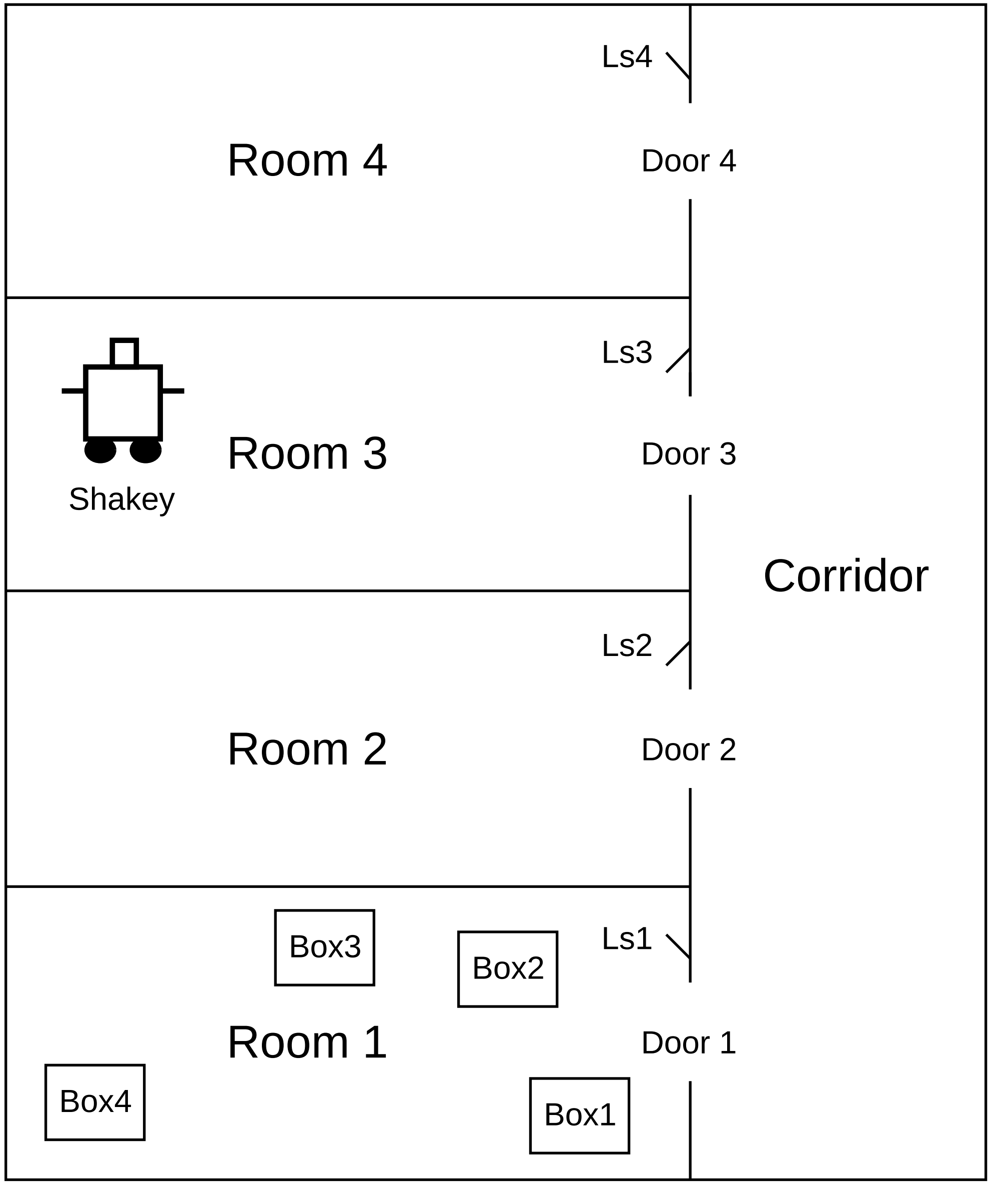


[Note: this abstracts away many important details!]

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

Shakey Example



Shakey Example, con't.

ACTION: Go(x,y):

PRECOND:

EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):

PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)

EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):

PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)

EFFECT: At(Shakey,y) $\wedge \neg$(At(Shakey,x))

Shakey Example, con't.

ACTION: Go(x,y):

PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)

EFFECT: At(Shakey,y) $\wedge \neg$(At(Shakey,x))

ACTION: Push(b,x,y):

PRECOND:

EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):

PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)

EFFECT: At(Shakey,y) $\wedge\neg$(At(Shakey,x))

ACTION: Push(b,x,y):

PRECOND: At(Shakey,x) $\wedge$ Pushable(b)

EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):
PRECOND: $At(Shakey,x) \wedge In(x,r) \wedge In(y,r)$
EFFECT: $At(Shakey,y) \wedge \neg(At(Shakey,x))$

ACTION: Push(b,x,y):
PRECOND: $At(Shakey,x) \wedge Pushable(b)$
EFFECT: $At(b,y) \wedge At(Shakey,y) \wedge \neg At(b,x) \wedge \neg At(Shakey,x)$

Shakey Example, con't.

ACTION: Go(x,y):

PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)

EFFECT: At(Shakey,y) $\wedge\neg$(At(Shakey,x))

ACTION: Push(b,x,y):

PRECOND: At(Shakey,x) $\wedge$ Pushable(b)

EFFECT: At(b,y) $\wedge$ At(Shakey,y) $\wedge\neg$At(b,x) $\wedge\neg$At(Shakey,x)

ACTION: ClimbUp(b):

PRECOND:

EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):

PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)

EFFECT: At(Shakey,y) $\wedge \neg$(At(Shakey,x))

ACTION: Push(b,x,y):

PRECOND: At(Shakey,x) $\wedge$ Pushable(b)

EFFECT: At(b,y) $\wedge$ At(Shakey,y) $\wedge \neg$At(b,x) $\wedge \neg$At(Shakey,x)

ACTION: ClimbUp(b):

PRECOND: At(Shakey,x) $\wedge$ At(b,x) $\wedge$ Climbable(b)

EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):
    PRECOND: At(Shakey,x) $\land$ In(x,r) $\land$ In(y,r)
    EFFECT: At(Shakey,y) $\land \neg$(At(Shakey,x))

ACTION: Push(b,x,y):
    PRECOND: At(Shakey,x) $\land$ Pushable(b)
    EFFECT: At(b,y) $\land$ At(Shakey,y) $\land \neg$At(b,x) $\land \neg$At(Shakey,x)

ACTION: ClimbUp(b):
    PRECOND: At(Shakey,x) $\land$ At(b,x) $\land$ Climbable(b)
    EFFECT: On(Shakey,b) $\land \neg$On(Shakey,Floor)

Shakey Example, con't.

ACTION: Go(x,y):
    PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)
    EFFECT: At(Shakey,y) $\wedge \neg$(At(Shakey,x))

ACTION: Push(b,x,y):
    PRECOND: At(Shakey,x) $\wedge$ Pushable(b)
    EFFECT: At(b,y) $\wedge$ At(Shakey,y) $\wedge \neg$At(b,x) $\wedge \neg$At(Shakey,x)

ACTION: ClimbUp(b):
    PRECOND: At(Shakey,x) $\wedge$ At(b,x) $\wedge$ Climbable(b)
    EFFECT: On(Shakey,b) $\wedge \neg$On(Shakey,Floor)

ACTION: ClimbDown(b):
    PRECOND:
    EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):

PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)

EFFECT: At(Shakey,y) $\wedge \neg$(At(Shakey,x))

ACTION: Push(b,x,y):

PRECOND: At(Shakey,x) $\wedge$ Pushable(b)

EFFECT: At(b,y) $\wedge$ At(Shakey,y) $\wedge \neg$At(b,x) $\wedge \neg$At(Shakey,x)

ACTION: ClimbUp(b):

PRECOND: At(Shakey,x) $\wedge$ At(b,x) $\wedge$ Climbable(b)

EFFECT: On(Shakey,b) $\wedge \neg$On(Shakey,Floor)

ACTION: ClimbDown(b):

PRECOND: On(Shakey,b)

EFFECT:

Shakey Example, con't.

ACTION: Go(x,y):
PRECOND: At(Shakey,x) $\wedge$ In(x,r) $\wedge$ In(y,r)
EFFECT: At(Shakey,y) $\wedge\neg$(At(Shakey,x))

ACTION: Push(b,x,y):
PRECOND: At(Shakey,x) $\wedge$ Pushable(b)
EFFECT: At(b,y) $\wedge$ At(Shakey,y) $\wedge\neg$At(b,x) $\wedge\neg$At(Shakey,x)

ACTION: ClimbUp(b):
PRECOND: At(Shakey,x) $\wedge$ At(b,x) $\wedge$ Climbable(b)
EFFECT: On(Shakey,b) $\wedge\neg$On(Shakey,Floor)

ACTION: ClimbDown(b):
PRECOND: On(Shakey,b)
EFFECT: On(Shakey,Floor) $\wedge\neg$ On(Shakey,b)

Shakey Example, con't.

ACTION: TurnOn(l):

PRECOND:

EFFECT:

Shakey Example, con't.

ACTION: TurnOn(l):

PRECOND: On(Shakey,b) $\wedge$ At(Shakey,x) $\wedge$ At(l,x)

EFFECT:

Shakey Example, con't.

ACTION: TurnOn(l):

PRECOND: On(Shakey,b) $\wedge$ At(Shakey,x) $\wedge$ At(l,x)

EFFECT: TurnedOn(l)

Shakey Example, con't.

ACTION: TurnOn(l):

PRECOND: On(Shakey,b) $\wedge$ At(Shakey,x) $\wedge$ At(l,x)

EFFECT: TurnedOn(l)

ACTION: TurnOff(l):

PRECOND:

EFFECT:

Shakey Example, con't.

ACTION: TurnOn(l):

PRECOND: On(Shakey,b) $\wedge$ At(Shakey,x) $\wedge$ At(l,x)

EFFECT: TurnedOn(l)

ACTION: TurnOff(l):

PRECOND: On(Shakey,b) $\wedge$ At(Shakey,x) $\wedge$ At(l,x)

EFFECT:

Shakey Example, con't.

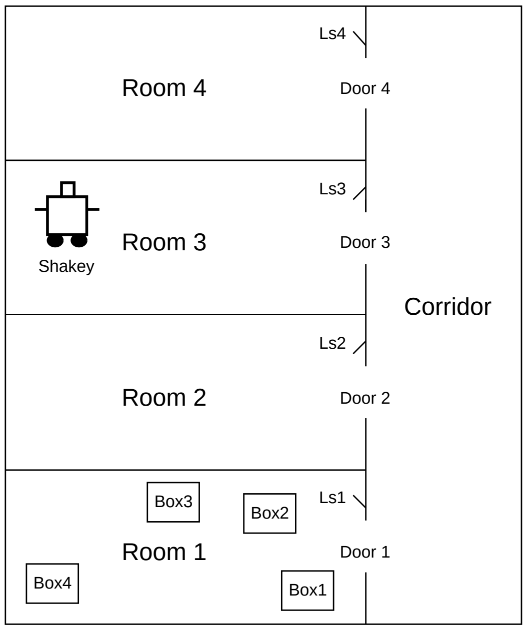
ACTION: TurnOn(l):
PRECOND: On(Shakey,b) $\wedge$ At(Shakey,x) $\wedge$ At(l,x)
EFFECT: TurnedOn(l)

ACTION: TurnOff(l):
PRECOND: On(Shakey,b) $\wedge$ At(Shakey,x) $\wedge$ At(l,x)
EFFECT: $\neg$TurnedOn(l)

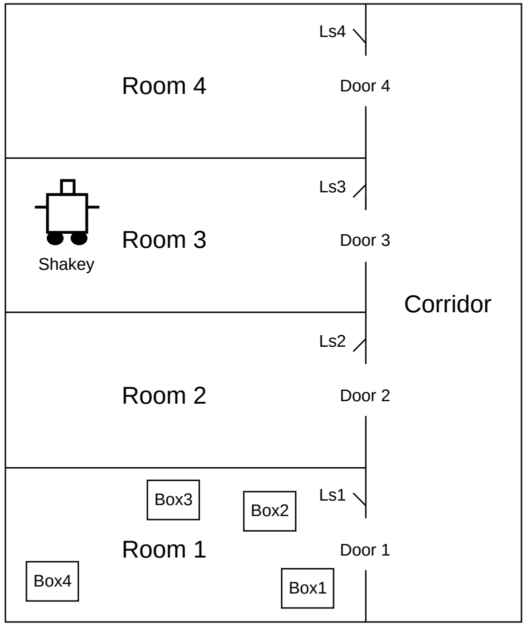
Shakey Example, con't.

INITIAL STATE:

In(...) Climbable(...) Pushable(...) At(...) TurnedOn(...)



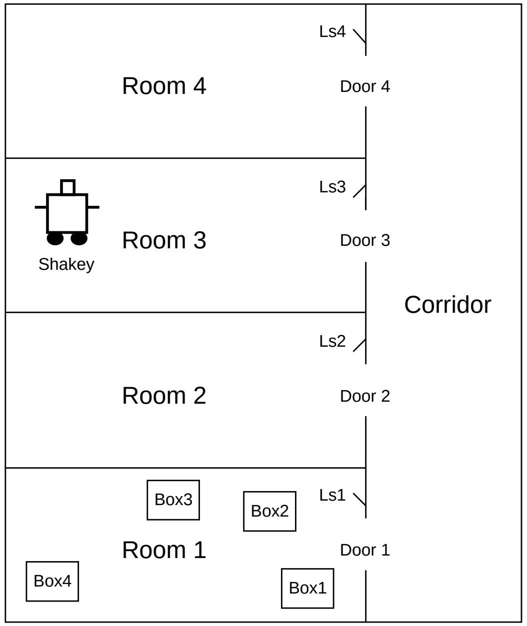
Shakey Example, con't.



INITIAL STATE:

In(Switch1,Room1) $\wedge$ In(Door1,Room1) $\wedge$ In(Door1,Corridor)
In(Switch1,Room2) $\wedge$ In(Door2,Room2) $\wedge$ In(Door2,Corridor)
In(Switch1,Room3) $\wedge$ In(Door3,Room3) $\wedge$ In(Door3,Corridor)
In(Switch1,Room4) $\wedge$ In(Door4,Room4) $\wedge$ In(Door4,Corridor)
In(Shakey,Room3) $\wedge$ At(Shakey,XS)
In(Box1,Room1) $\wedge$ In(Box2,Room1) $\wedge$ In(Box3,Room1) $\wedge$ In(Box4,Room1)

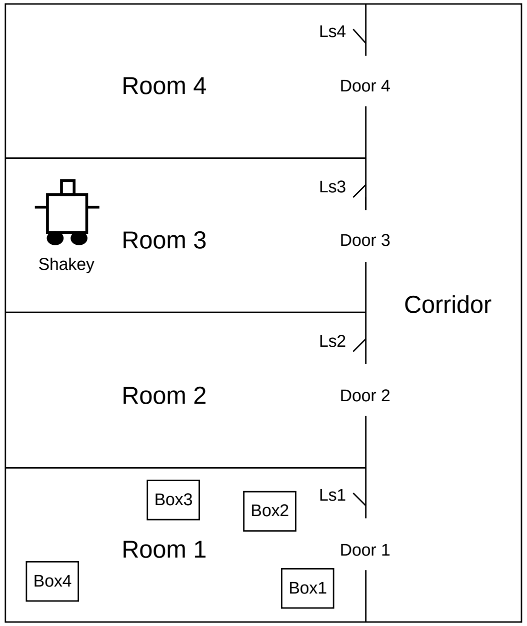
Shakey Example, con't.



INITIAL STATE (con't.):

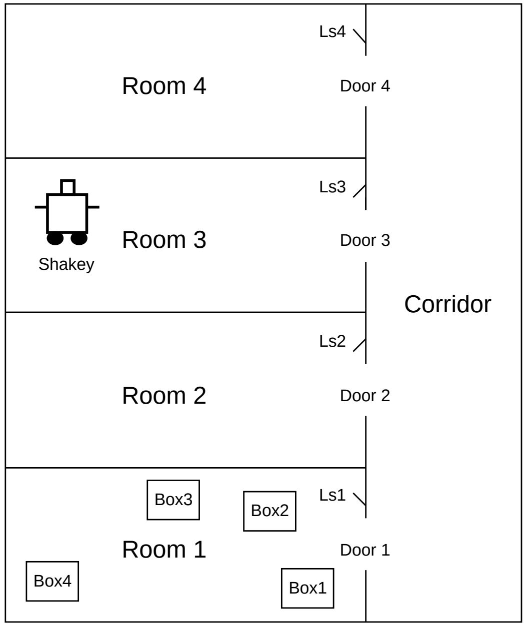
Climbable(Box1) ∧ Climbable(Box2) ∧ Climbable(Box3) ∧ Climbable(Box4)
Pushable(Box1) ∧ Pushable(Box2) ∧ Pushable(Box3) ∧ Pushable(Box4)
At(Box1, X1) ∧ At(Box2, X2) ∧ At(Box3, X3) ∧ At(Box4, X4)
TurnedOn(Switch1) ∧ TurnedOn(Switch4)

Shakey Example, con't.



Plan to achieve goal of getting Box2 into Room2:

Shakey Example, con't.



Plan to achieve goal of getting Box2 into Room2:

Go(XS,Door3)
Go(Door3,Door1)
Go(Door1,X2)
Push(Box2, X2, Door1)
Push(Box2, Door1, Door2)
Push(Box2, Door2, Switch2)

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

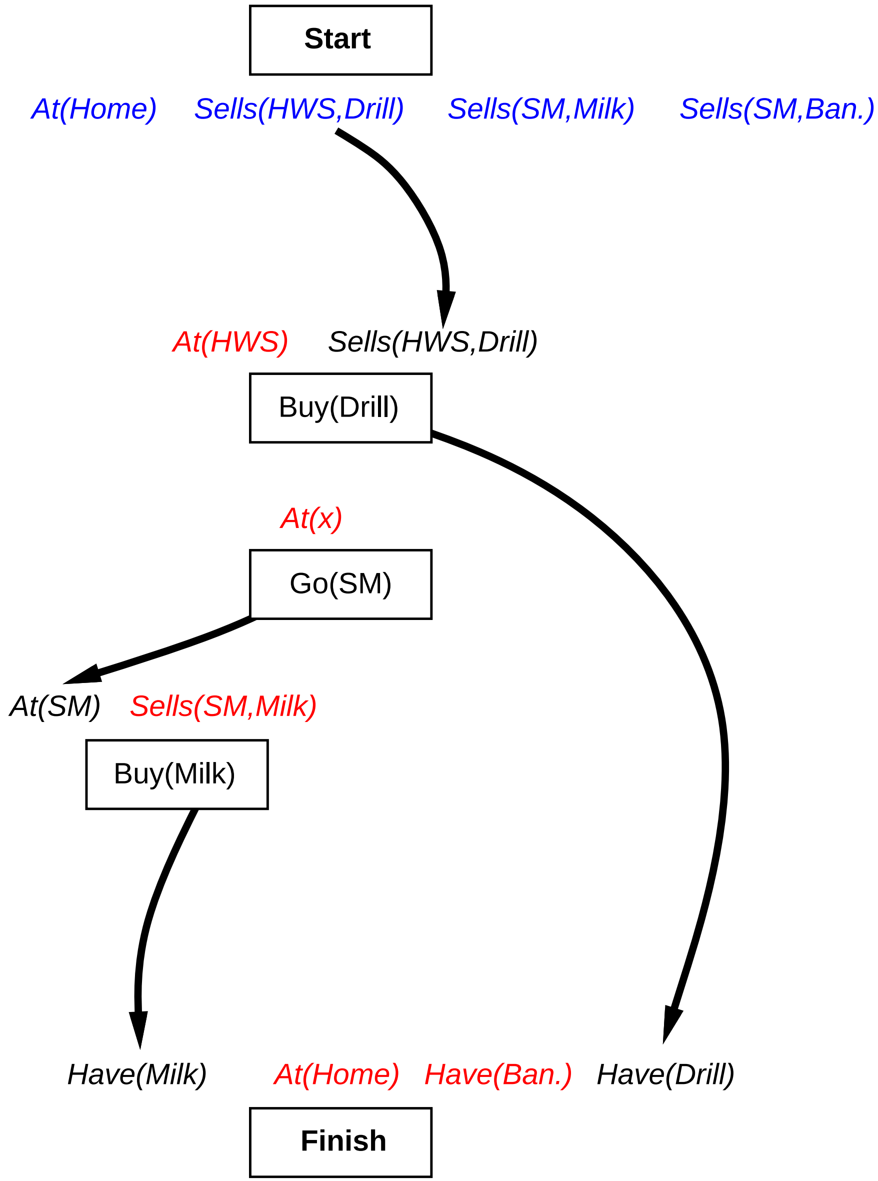
Start

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

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

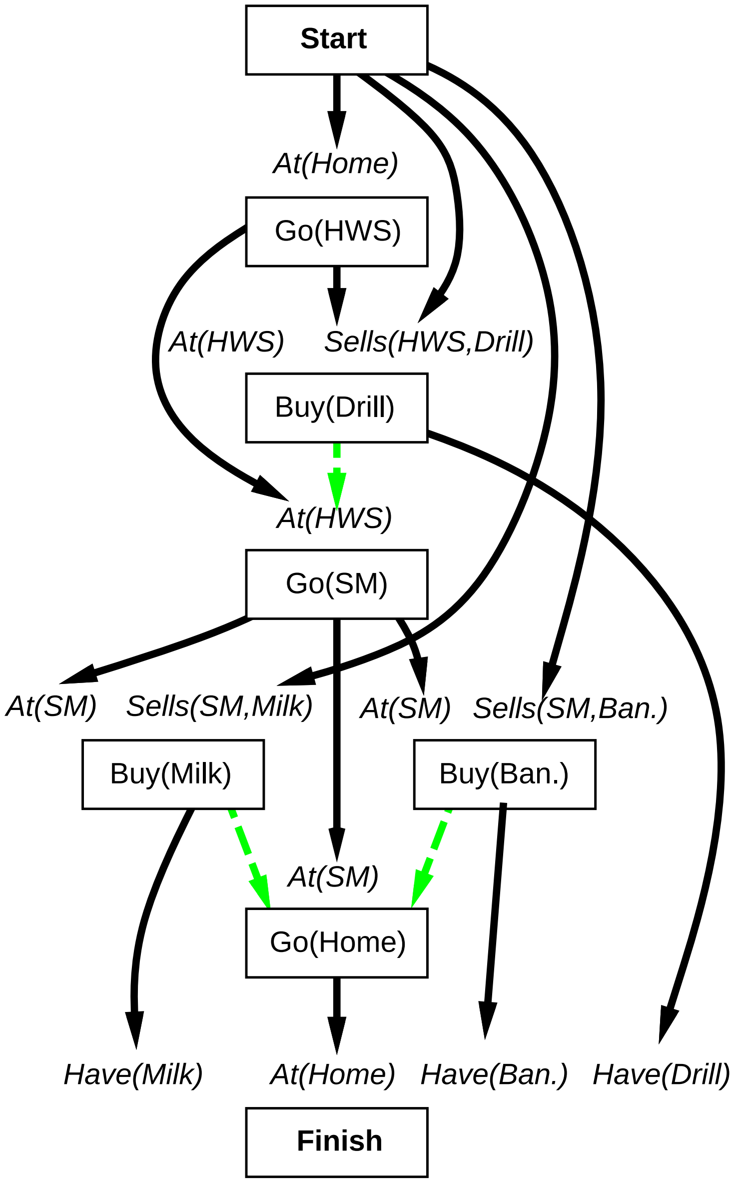
Finish

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($plan$)

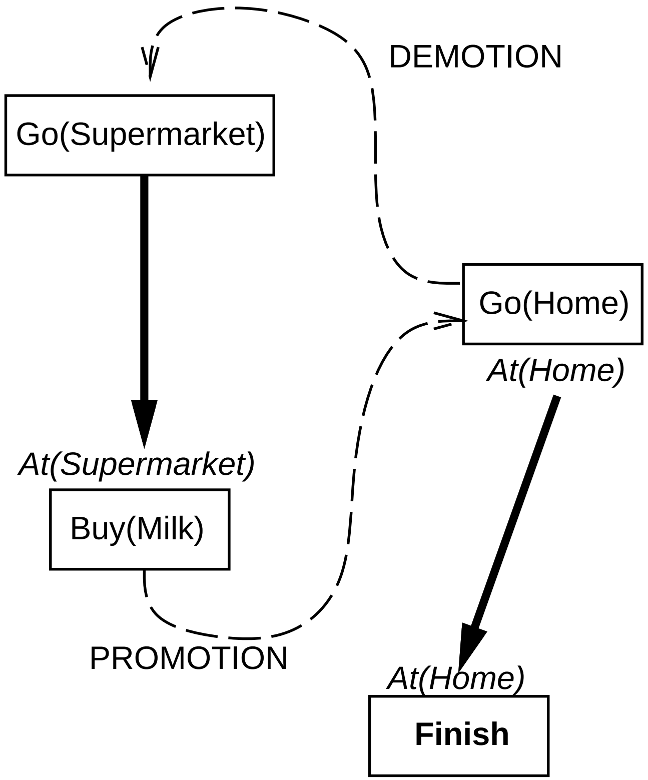
---

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)

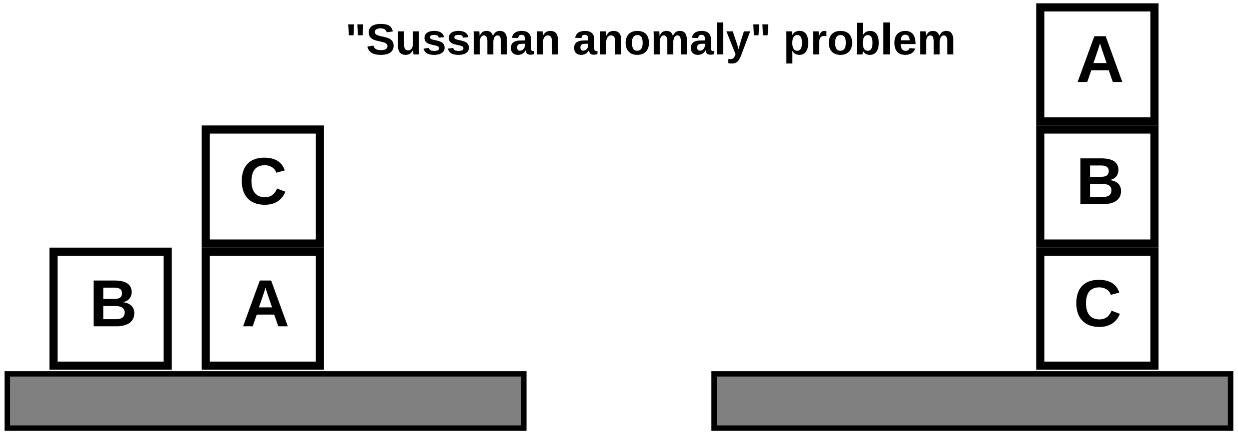
Extensions for disjunction, universals, negation, conditionals

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)

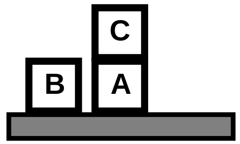
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Example contd.

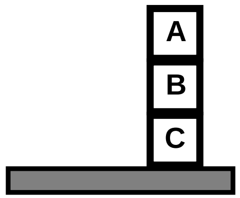
START

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

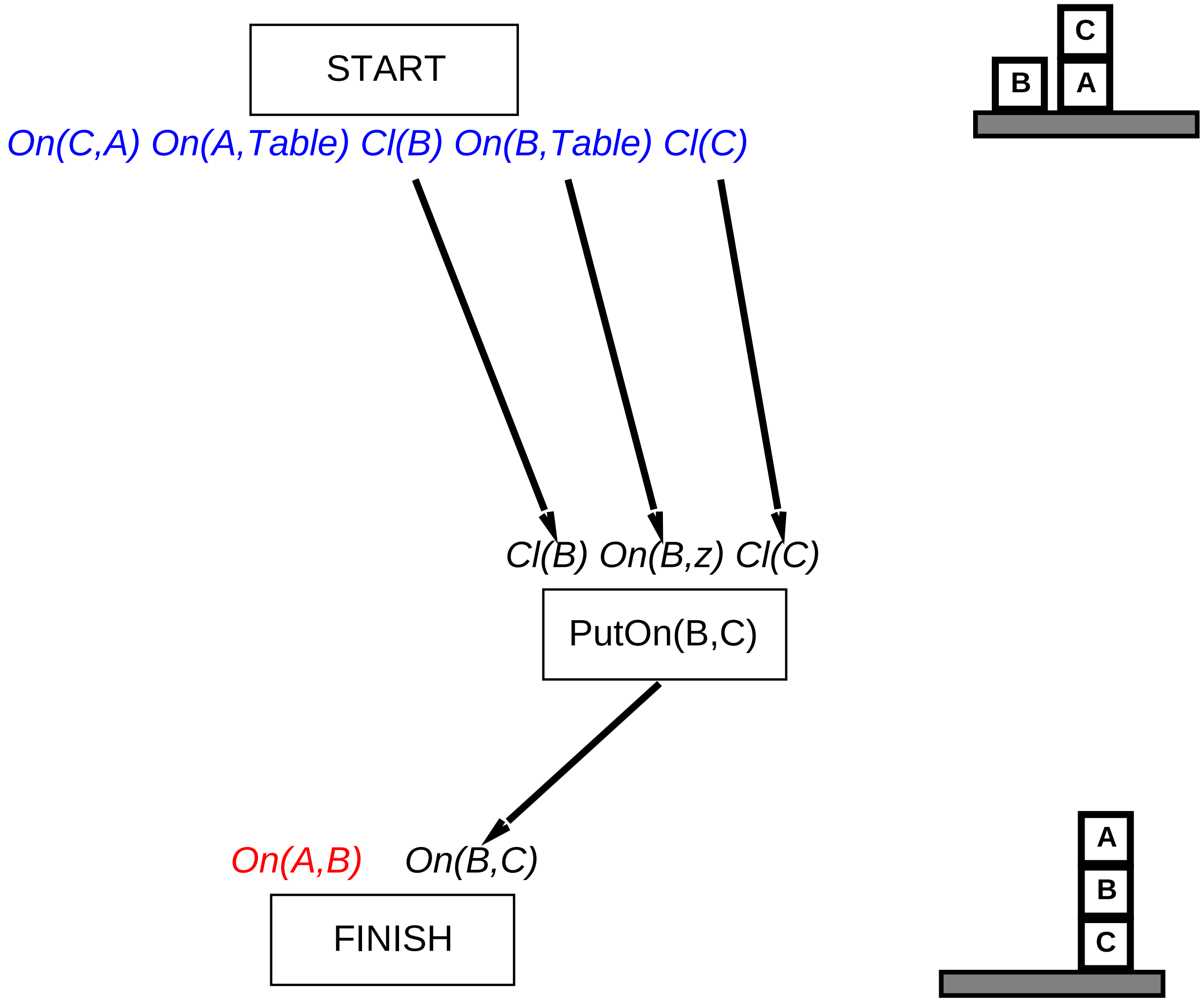


On(A,B)   On(B,C)

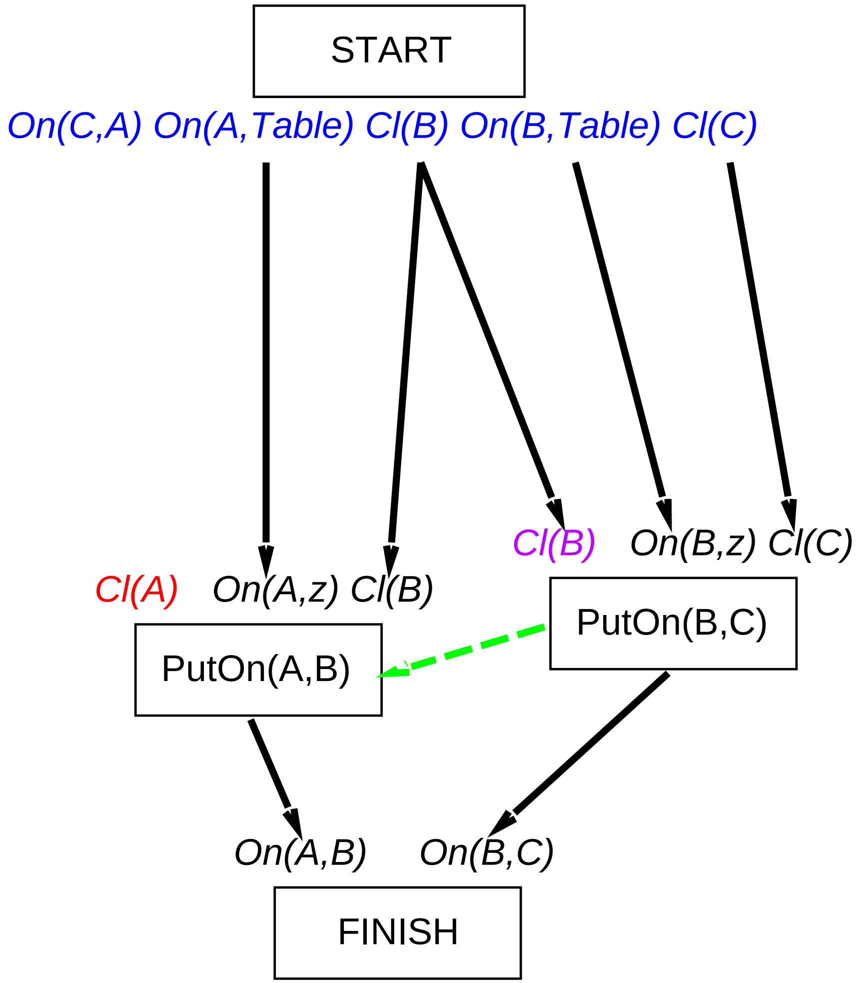
FINISH

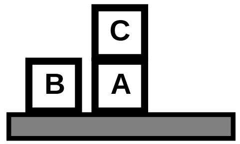


Example contd.

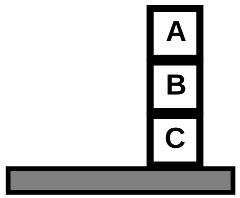


Example contd.

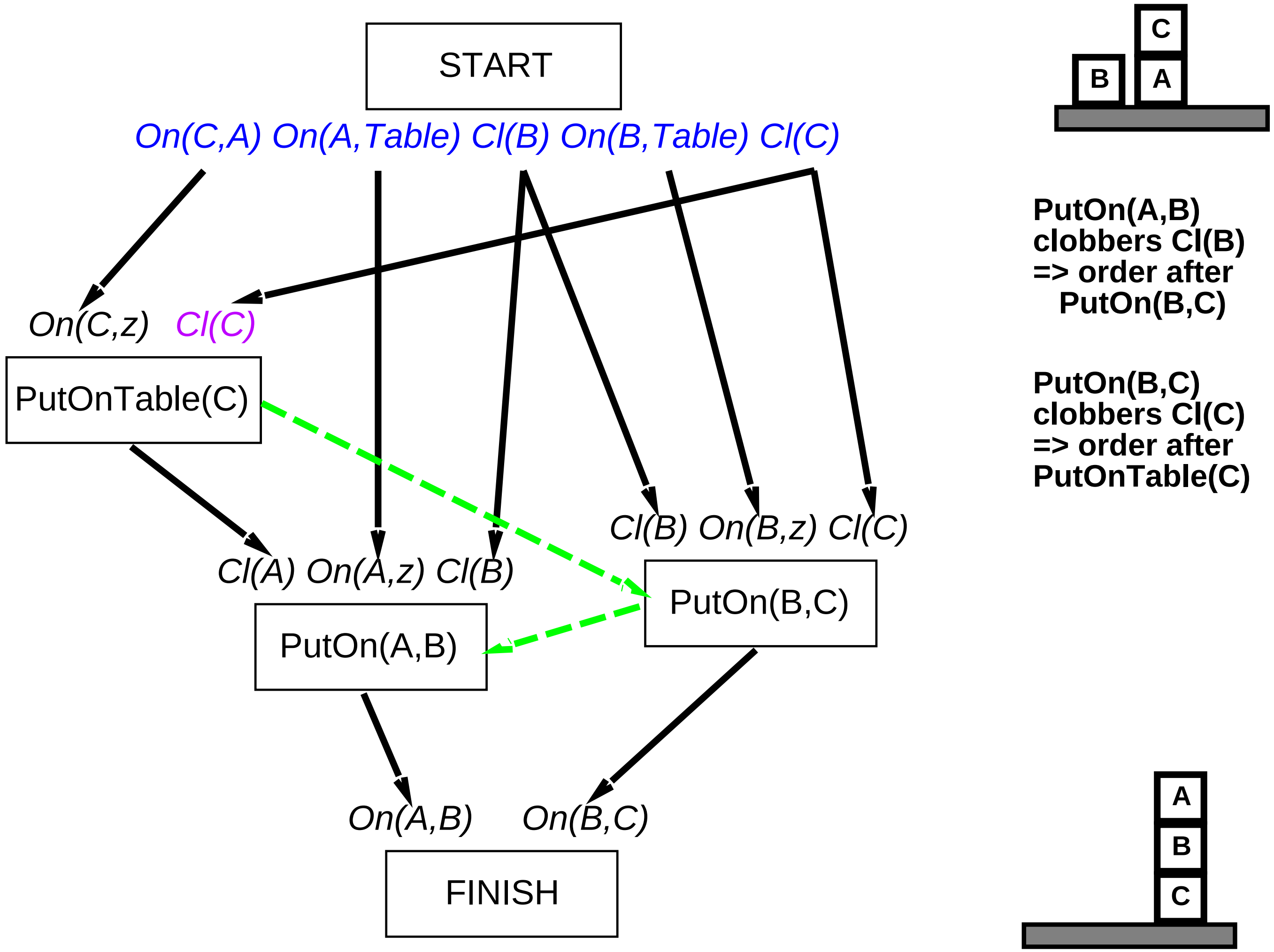




PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



Example contd.



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

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

Heuristics for Planning

Most obvious Heuristic: Number of distinct open preconditions.

Overestimates: When actions achieve multiple goals

Underestimates: When negative interactions between plan steps

Better way: Use planning graph for generating better heuristic estimates.

Planning Graphs

Levels: Correspond to time steps in the plan ($0 =$ initial state)

Each level contains literals $+$ actions: those that *could* be true or executed

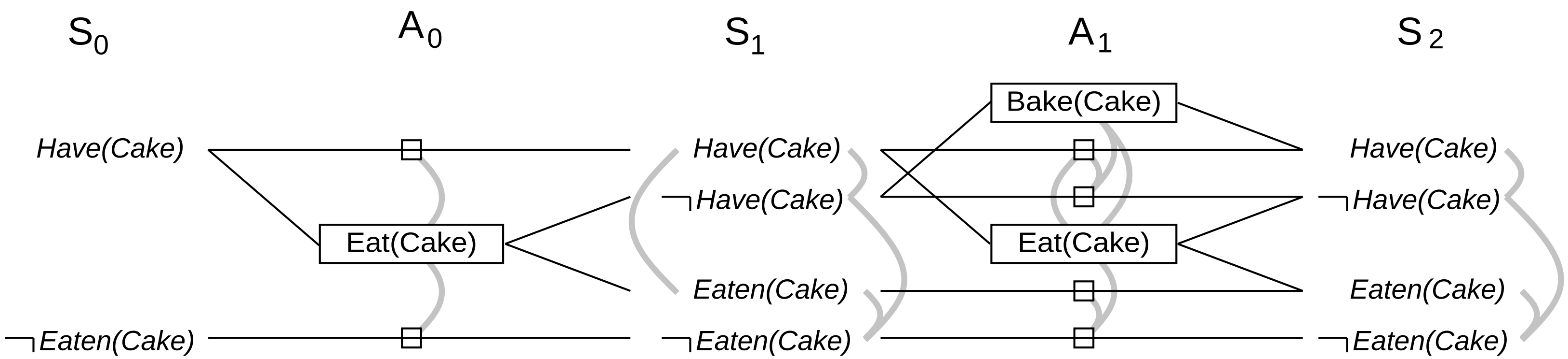
Number of planning steps in planning graph is good estimate of how difficult it is to acheive a given literal from initial state

Can be constructed very efficiently

Works only for *propositionalized problems*

Planning Graph – Have Cake

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  Precond: $Have(Cake)$
  Effect: $\neg Have(Cake) \wedge Eaten(Cake)$)
$Action(Bake(Cake)$
  Precond: $\neg Have(Cake)$
  Effect: $Have(Cake))$



Persistence actions   Mutual exclusion (mutex) links

Mutex Links

A mutex relation holds between two actions at a given level if any of the following is true:

♢ Inconsistent effects: one action negates another.

♢ Interference: one of effects of action is negation of precondition of another action.

♢ Competing needs: one of preconditions of action is mutually exclusive with precondition of other.

A mutex relation holds between two literals at a given level if:

♢ One is negation of other.

♢ Each possible pair of actions that could achieve the literals is mutex.

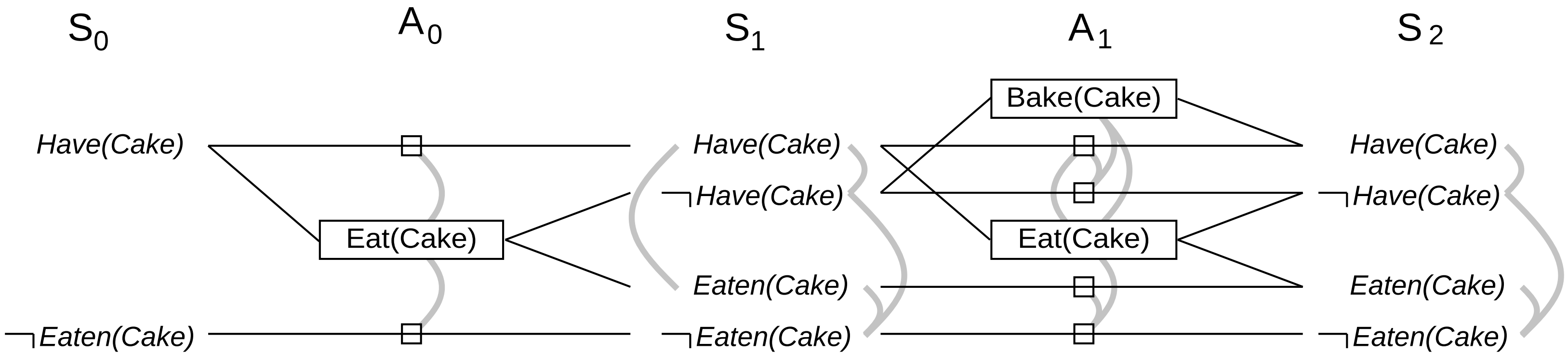
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- ♢ Max-level: Maximum level cost of any goal
- ♢ Level sum: Sum of level costs of goals (note: inadmissible)
- ♢ Set-level: Level at which all literals appear without mutex



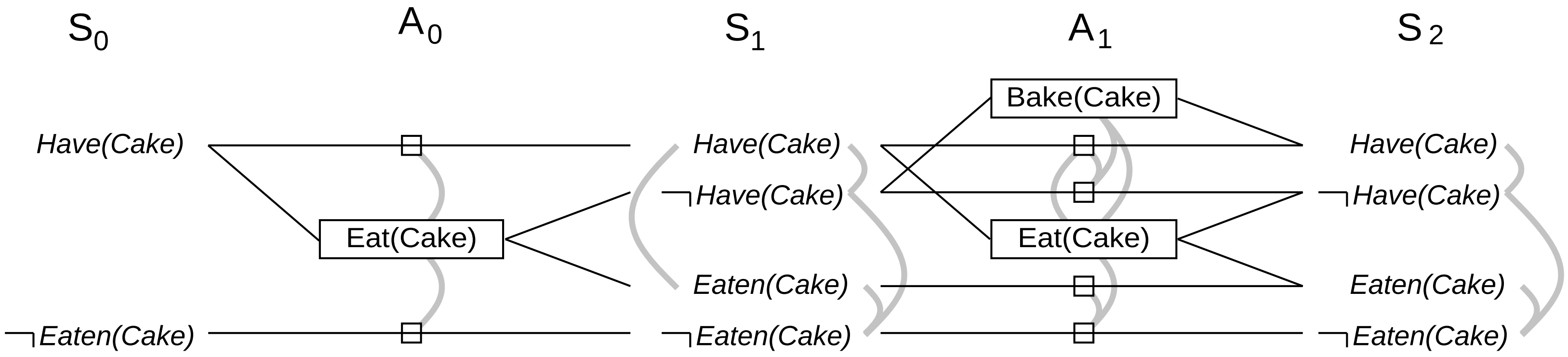
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- ◇ Max-level: Maximum level cost of any goal
- ◇ Level sum: Sum of level costs of goals (note: inadmissible)
- ◇ Set-level: Level at which all literals appear without mutex



Max-level cost?

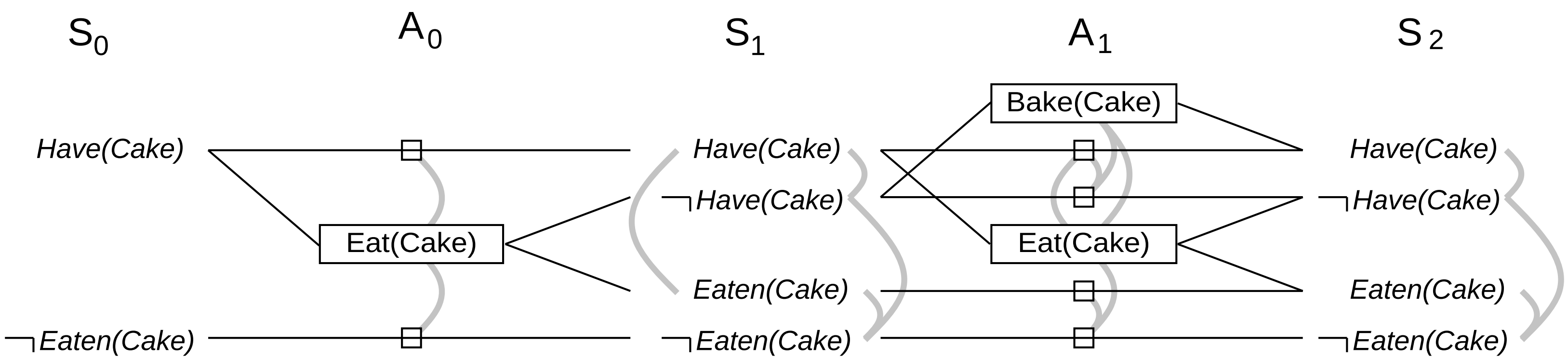
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- ◇ Max-level: Maximum level cost of any goal
- ◇ Level sum: Sum of level costs of goals (note: inadmissible)
- ◇ Set-level: Level at which all literals appear without mutex



Max-level cost? 1

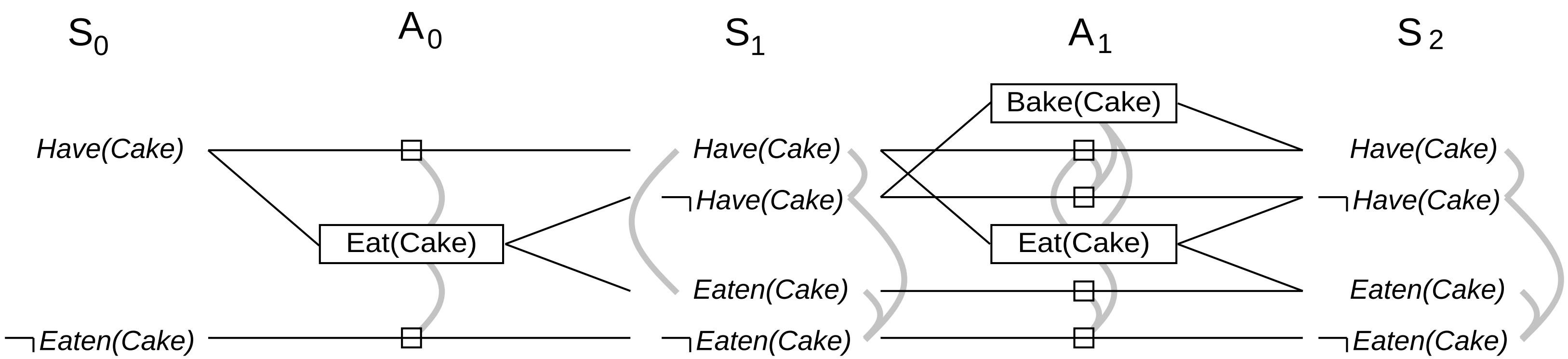
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- ♢ Max-level: Maximum level cost of any goal
- ♢ Level sum: Sum of level costs of goals (note: inadmissible)
- ♢ Set-level: Level at which all literals appear without mutex



Max-level cost? 1 Level sum cost?

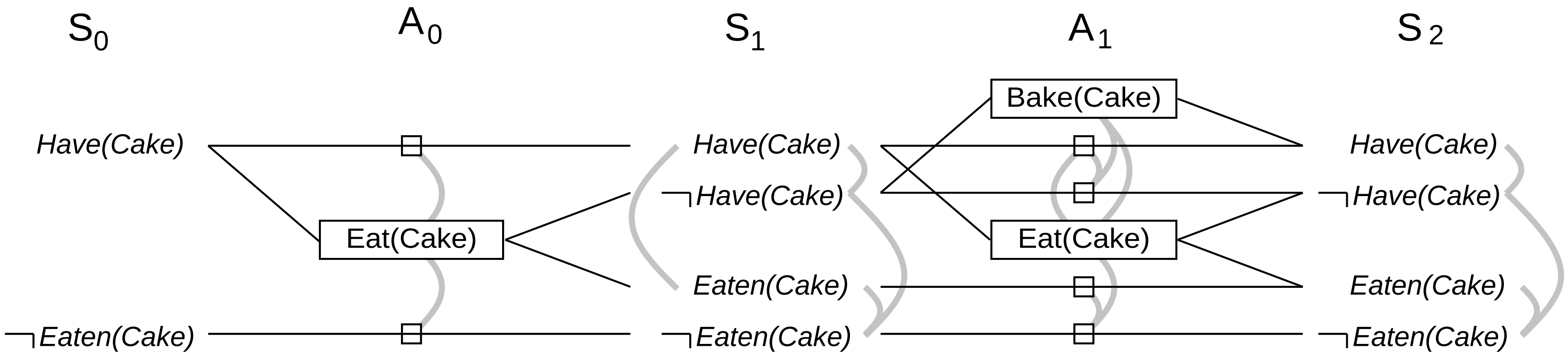
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- ◊ Max-level: Maximum level cost of any goal
- ◊ Level sum: Sum of level costs of goals (note: inadmissible)
- ◊ Set-level: Level at which all literals appear without mutex



Max-level cost? 1 Level sum cost? 1

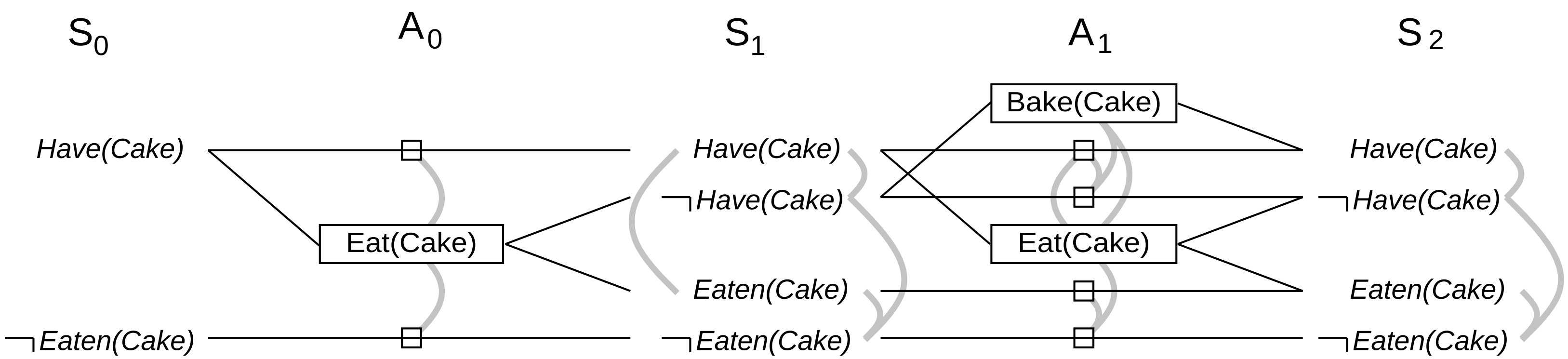
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- ◇ Max-level: Maximum level cost of any goal
- ◇ Level sum: Sum of level costs of goals (note: inadmissible)
- ◇ Set-level: Level at which all literals appear without mutex



Max-level cost? 1 Level sum cost? 1 Set-level Cost?

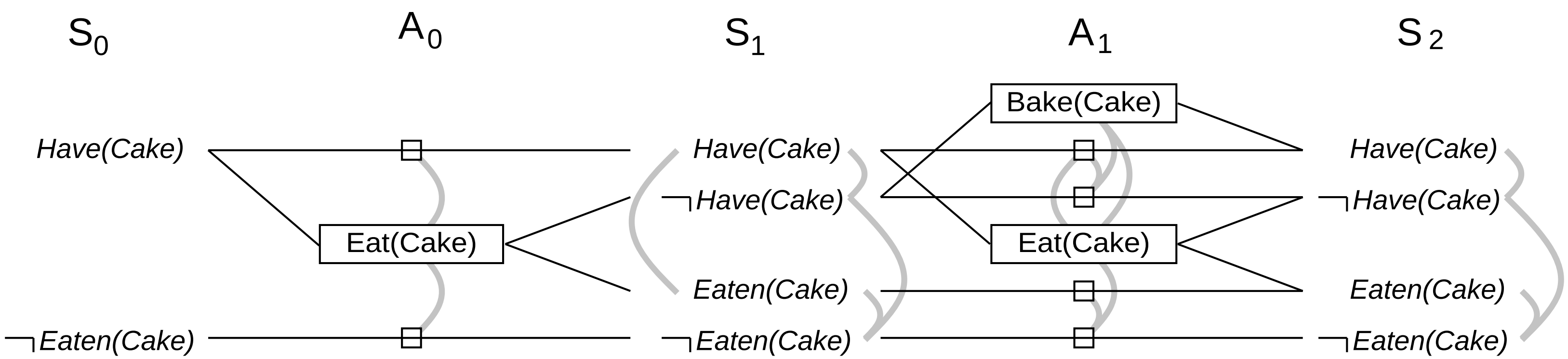
Heuristics from Planning Graphs

Estimate cost of goal literal = level it first appears = Level Cost

Use serial planning graphs to allow only one action at a time.

Cost of conjunction of goals:

- ♢ Max-level: Maximum level cost of any goal
- ♢ Level sum: Sum of level costs of goals (note: inadmissible)
- ♢ Set-level: Level at which all literals appear without mutex



Max-level cost? 1 Level sum cost? 1 Set-level Cost? 2

GraphPlan algorithm

Extracting a plan from planning graph...

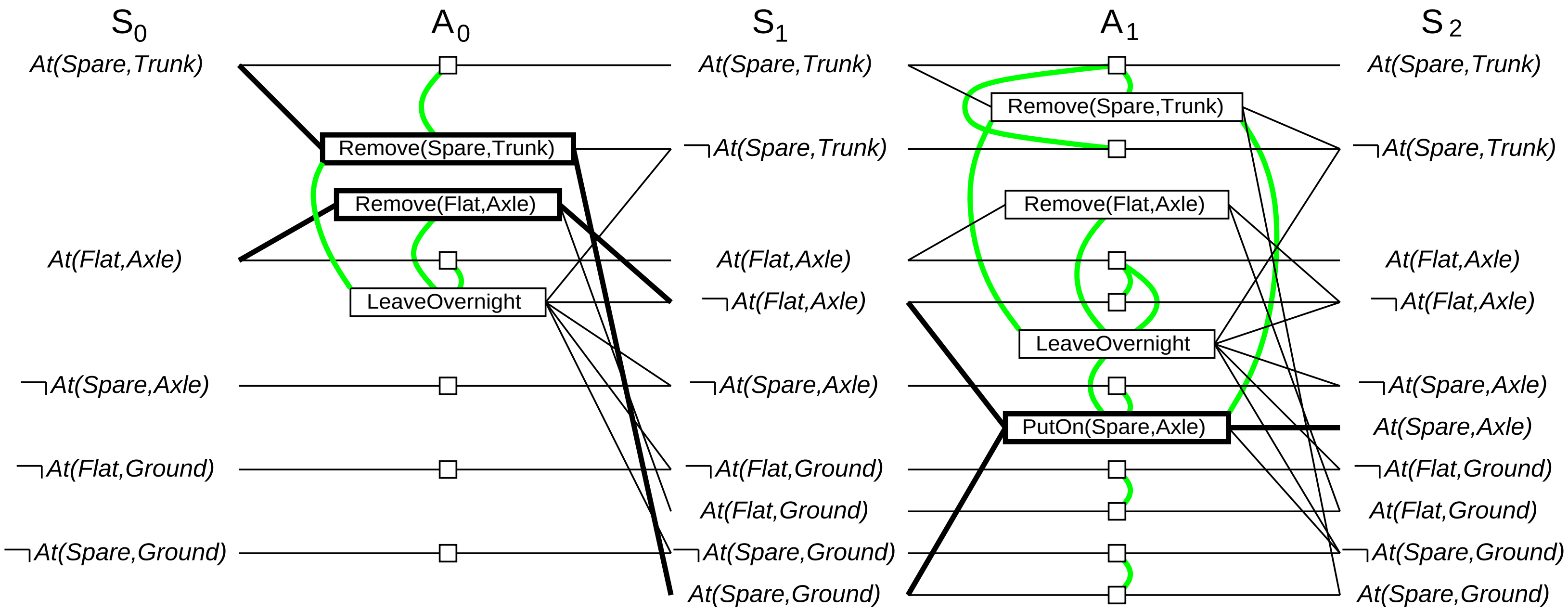
function GRAPHPLAN($problem$) **returns** solution or failure
  $graph \leftarrow$ Initial-Planning-Graph($problem$)
  $goals \leftarrow$ Goals[$problem$]
  loop do
    if $goals$ all non-mutex in last level of $graph$, **then do**
      $solution \leftarrow$ Extract-Solution($graph$, $goals$, Length($graph$))
      if $solution \neq failure$ **then return** $solution$
      else if No-Solution-Possible($graph$) **then return** $failure$
  $graph \leftarrow$ Expand-Graph($graph$, $problem$)

Spare Tire Problem

$Init(At(Flat,Axle) \land At(Spare,Trunk))$
$Goal(At(Spare,Axle))$
$Action(Remove(Spare,Trunk),$
    Precond: $At(Spare,Trunk)$
    Effect: $\neg At(Spare,Trunk) \land At(Spare,Ground))$
$Action(Remove(Flat,Axle),$
    Precond: $At(Flat,Axle)$
    Effect: $\neg At(Flat,Axle) \land At(Flat,Ground))$
$Action(PutOn(Spare,Axle),$
    Precond: $At(Spare,Ground) \land \neg At(Flat,Axle)$
    Effect: $\neg At(Spare,Ground) \land At(Spare,Axle))$
$Action(LeaveOvernight,$
    Precond:
    Effect: $\neg At(Spare,Ground) \land \neg At(Spare,Axle) \land \neg At(Spare,Trunk)$
        $\land \neg At(Flat,Ground) \land \neg At(Flat,Axle))$

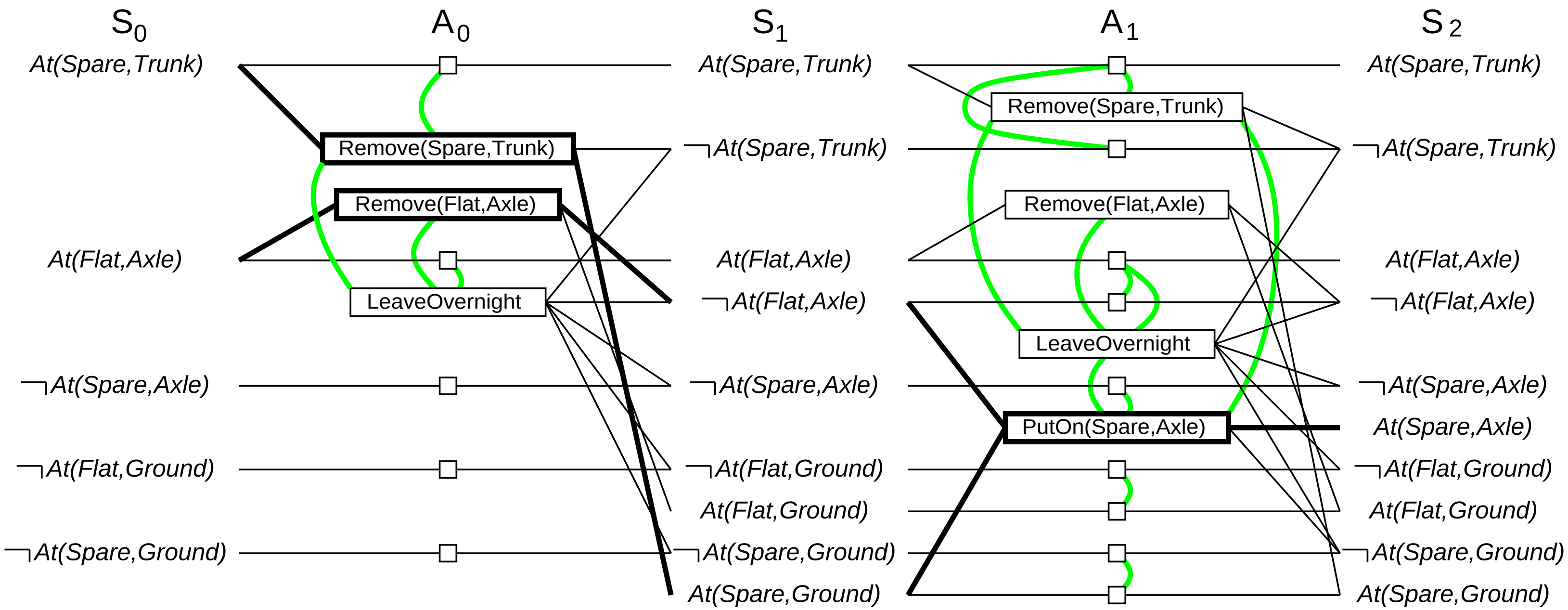
Planning Graph – Spare Tire

(Not all mutex's shown.)



Planning Graph – Spare Tire

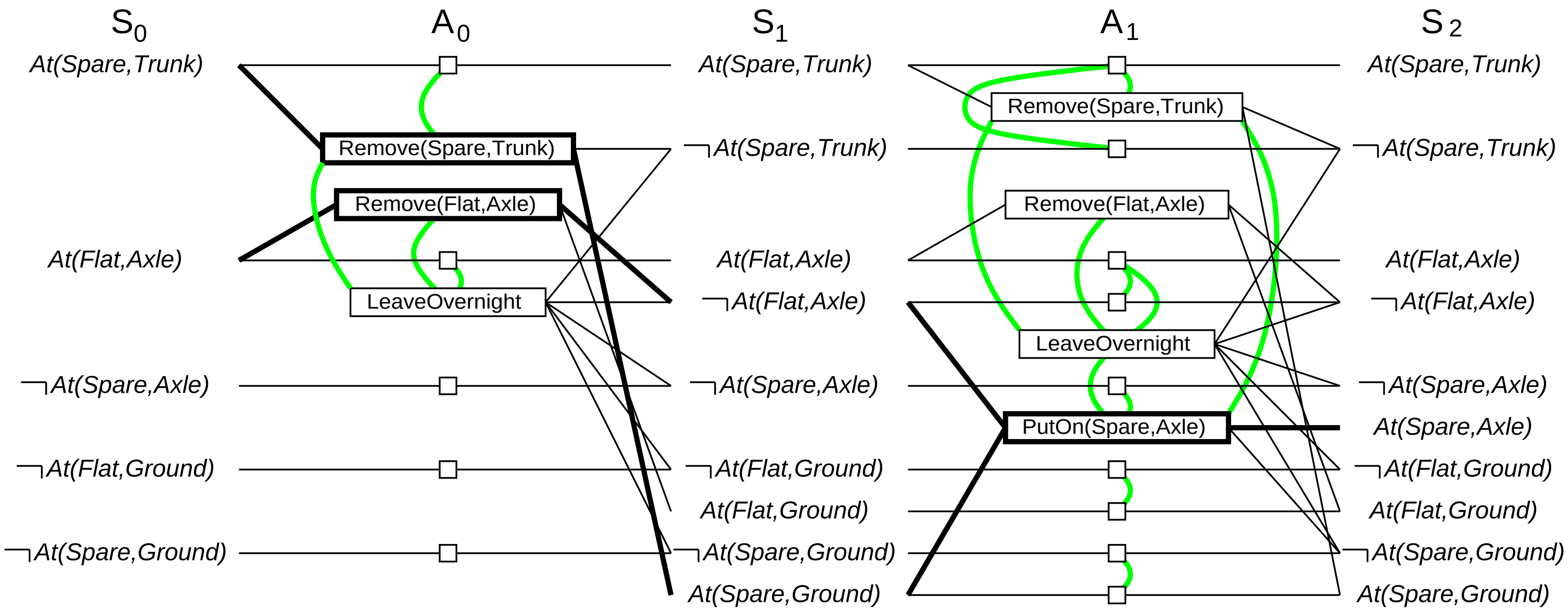
(Not all mutex's shown.)



Example of Inconsistent Effects?

Planning Graph – Spare Tire

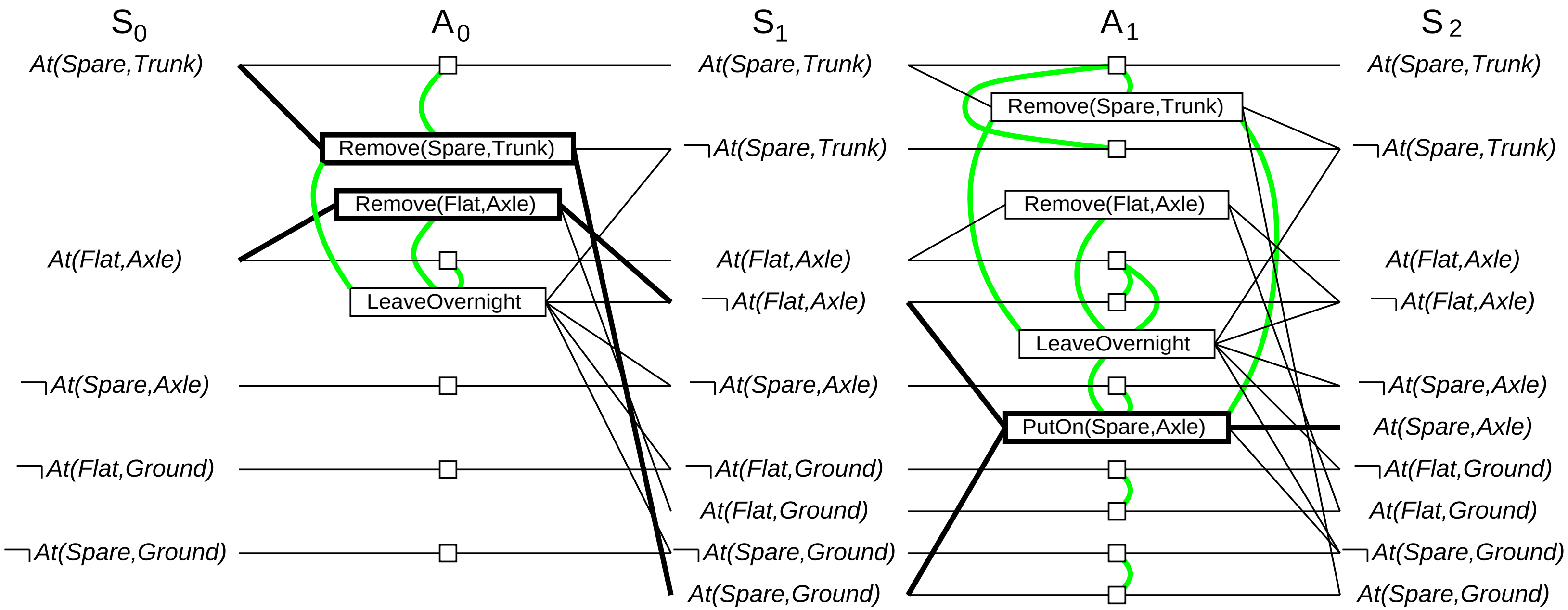
(Not all mutex's shown.)



Example of Inconsistent Effects? Remove(Spare,Trunk) and LeaveOvernight

Planning Graph – Spare Tire

(Not all mutex's shown.)

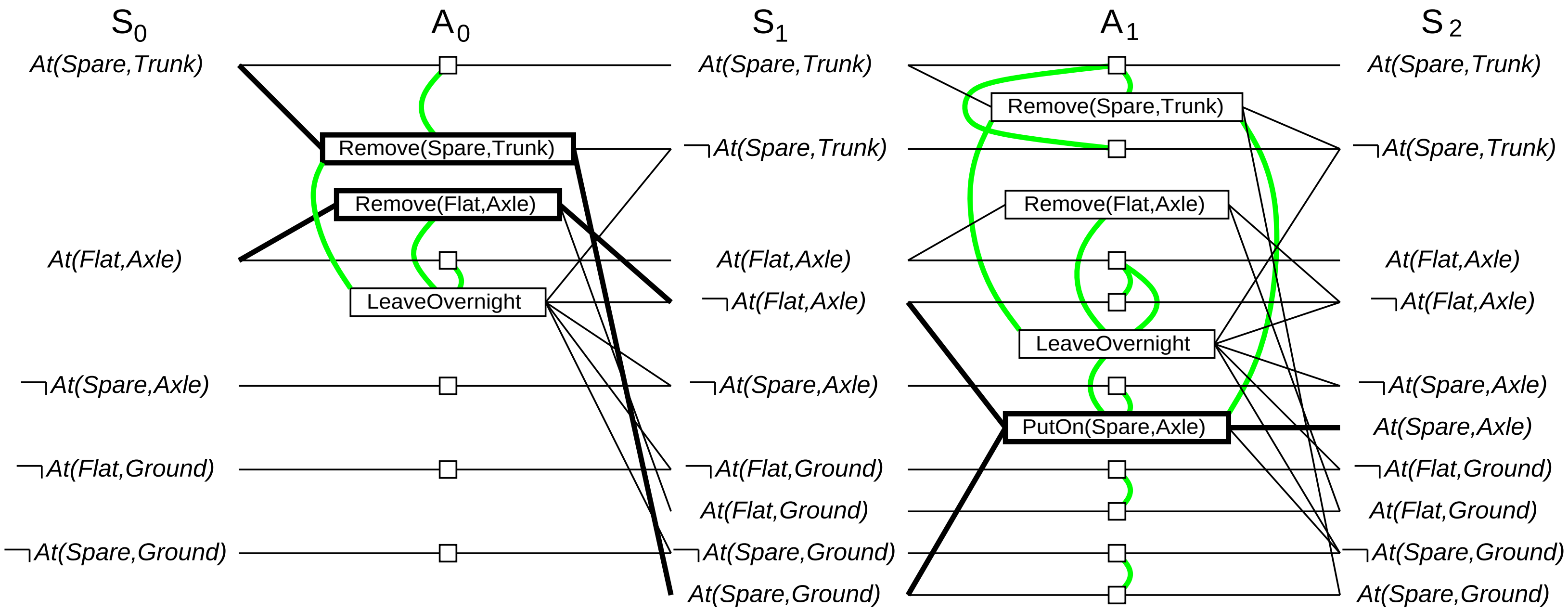


Example of Inconsistent Effects? Remove(Spare,Trunk) and LeaveOvernight

Example of Interference?

Planning Graph – Spare Tire

(Not all mutex's shown.)

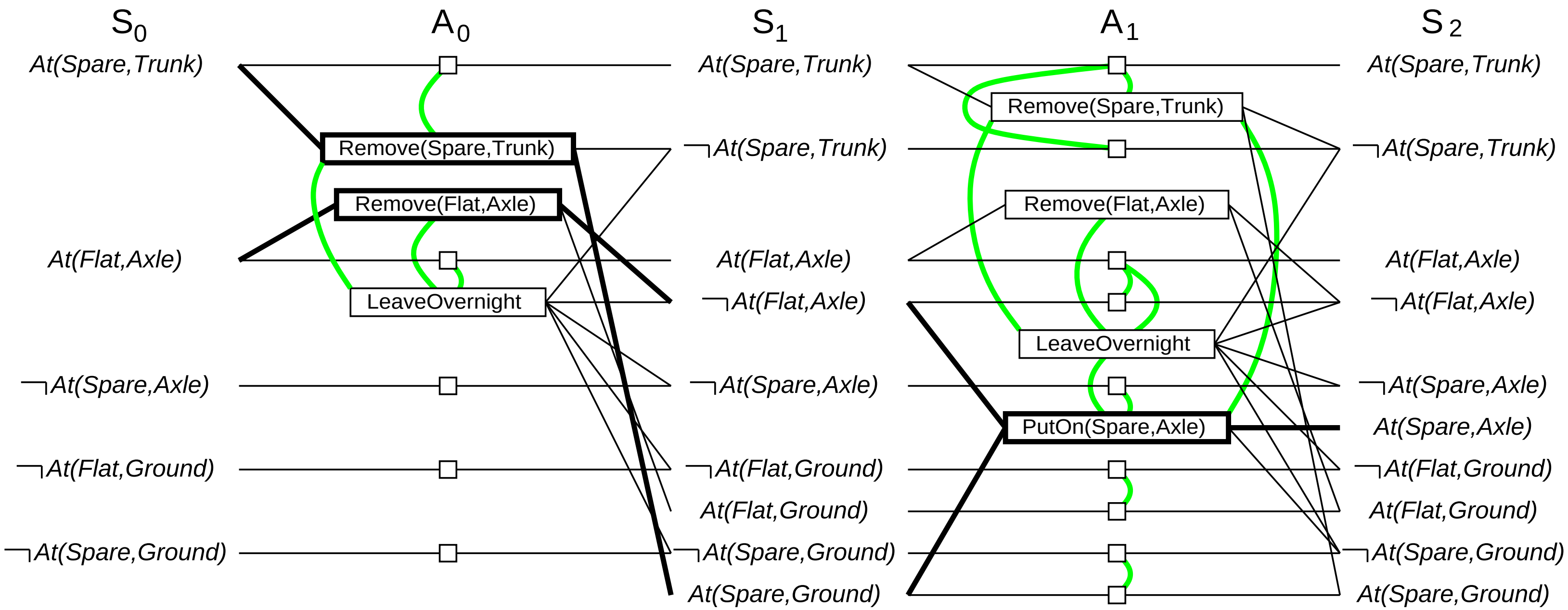


Example of Inconsistent Effects? Remove(Spare,Trunk) and LeaveOvernight

Example of Interference? Remove(Flat,Axle) LeaveOvernight

Planning Graph – Spare Tire

(Not all mutex's shown.)



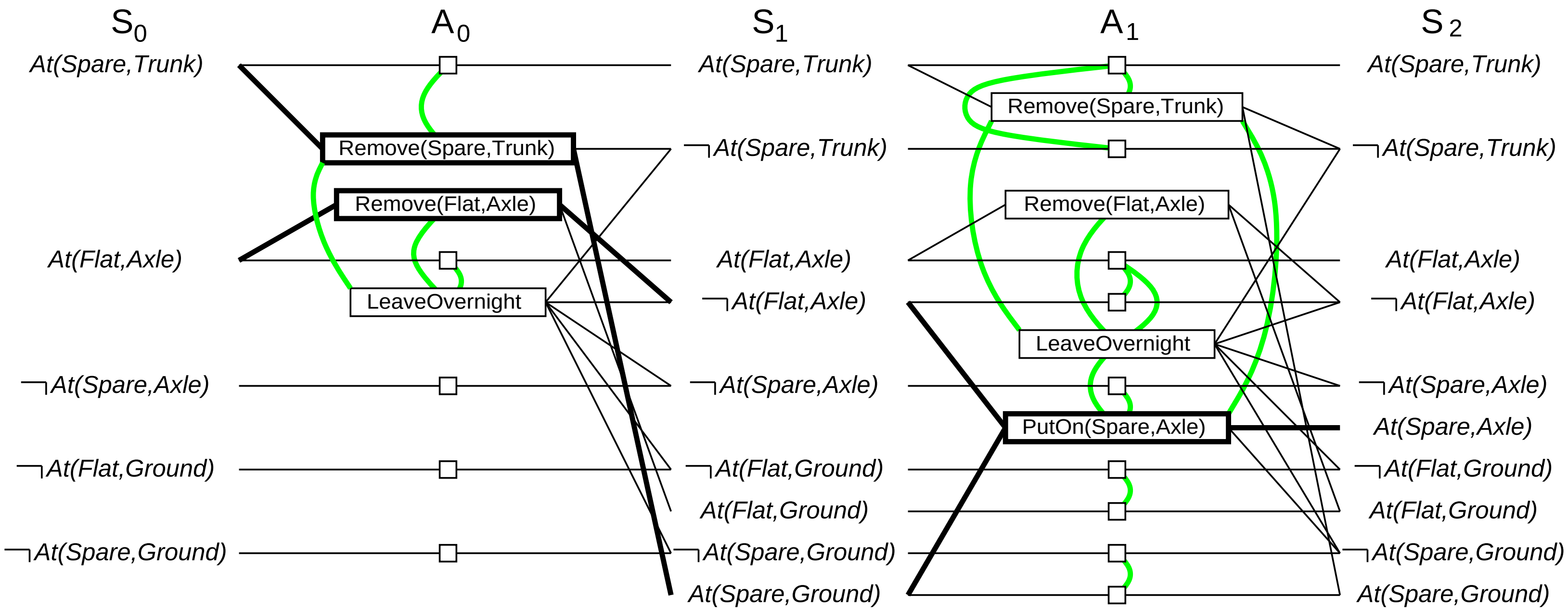
Example of Inconsistent Effects? Remove(Spare,Trunk) and LeaveOvernight

Example of Interference? Remove(Flat,Axle) LeaveOvernight

Example of Competing Needs?

Planning Graph – Spare Tire

(Not all mutex's shown.)



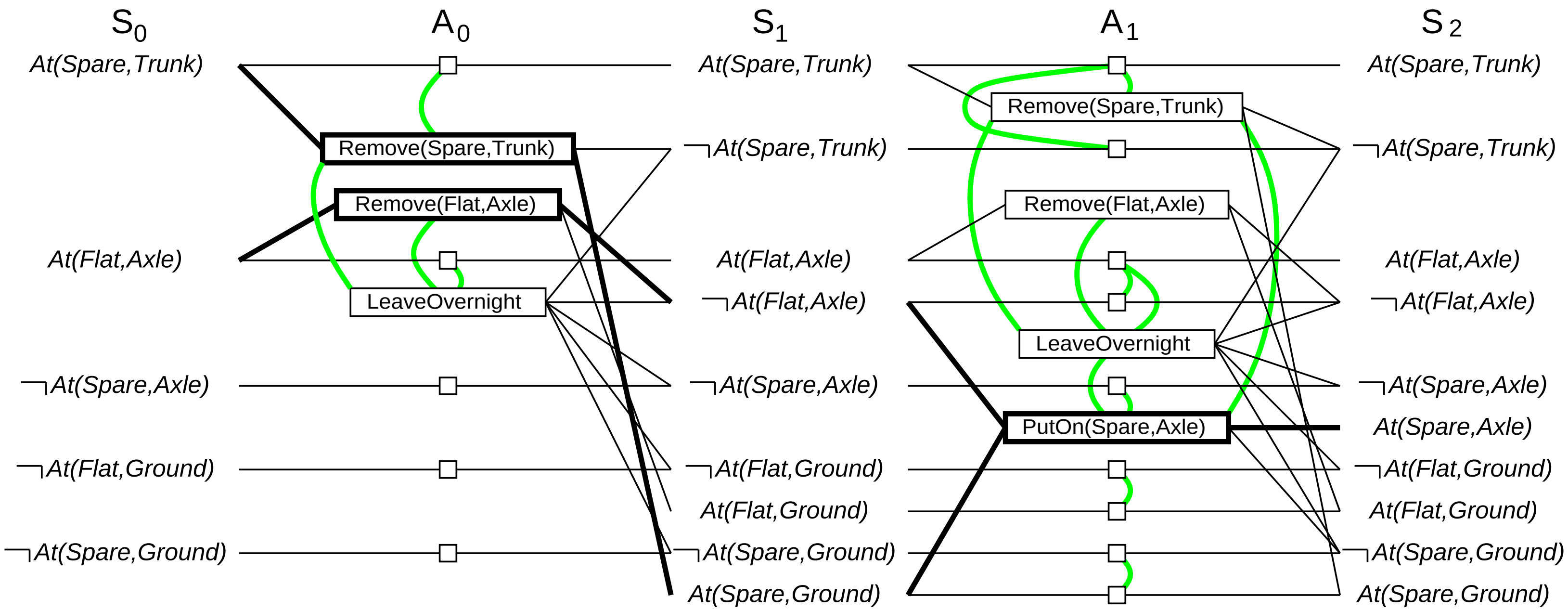
Example of Inconsistent Effects? Remove(Spare,Trunk) and LeaveOvernight

Example of Interference? Remove(Flat,Axle) and LeaveOvernight

Example of Competing Needs? PutOn(Spare,Axle) and Remove(Flat,Axle)

Planning Graph – Spare Tire

(Not all mutex's shown.)



Example of Inconsistent Effects? Remove(Spare,Trunk) and LeaveOvernight

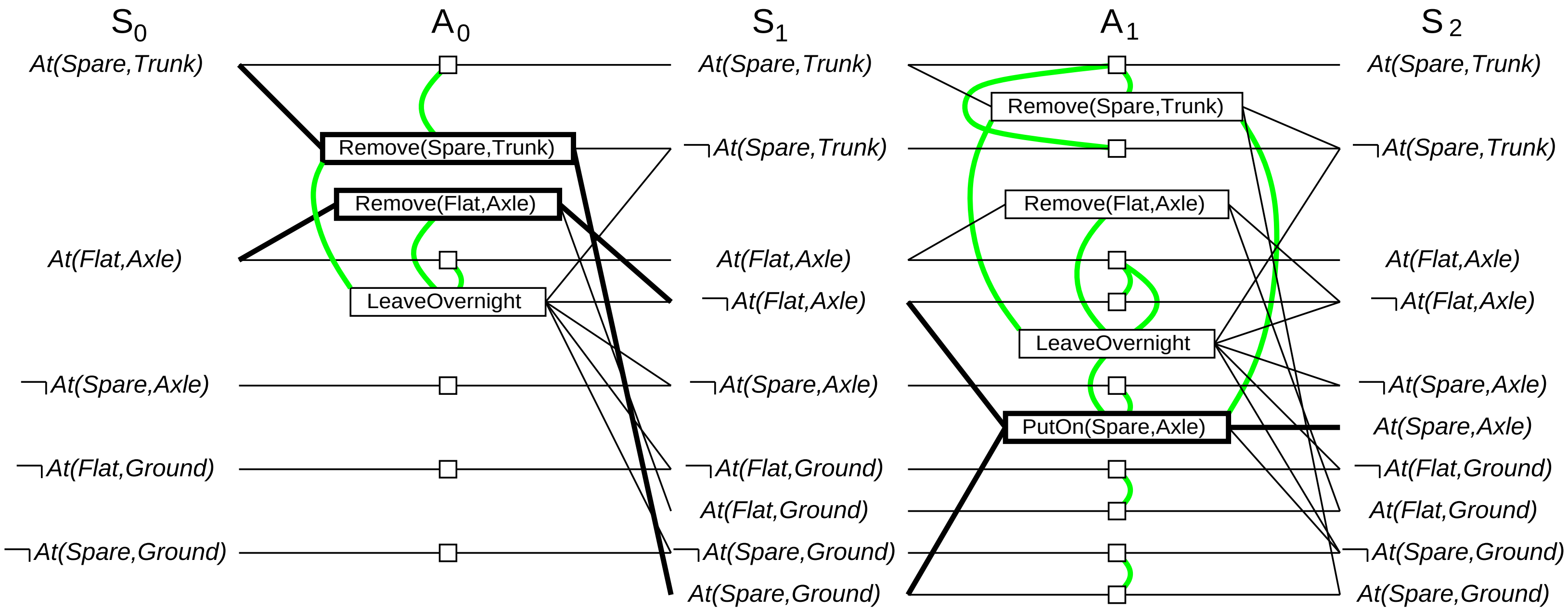
Example of Interference? Remove(Flat,Axle) and LeaveOvernight

Example of Competing Needs? PutOn(Spare,Axle) and Remove(Flat,Axle)

Example of Inconsistent Support?

Planning Graph – Spare Tire

(Not all mutex's shown.)



Example of Inconsistent Effects? Remove(Spare,Trunk) and LeaveOvernight

Example of Interference? Remove(Flat,Axle) and LeaveOvernight

Example of Competing Needs? PutOn(Spare,Axle) and Remove(Flat,Axle)

Example of Inconsistent Support? At(Spare,Axle) and At(Flat,Axle)

Summary of Planning Graphs

♢ Yield useful heuristics of state-space and partial order planners

♢ Consists of multiple layers of literals and actions that can occur at each time step

♢ Includes mutex relations to exclude co-occurrences

♢ Plan can be extracted directly from graph

Summary

♢ Planning systems operate on explicit representations of states and actions

♢ STRIPS language describes actions in terms of preconditions and effects.

♢ Partial-order planning (POP) algorithms explore space of plans without committing to a totally ordered sequence of actions.

♢ POP algorithms work backwards from goal, and are particularly effective on problems amenable to divide-and-conquer.

♢ No consensus on any specific planning approach being the best.