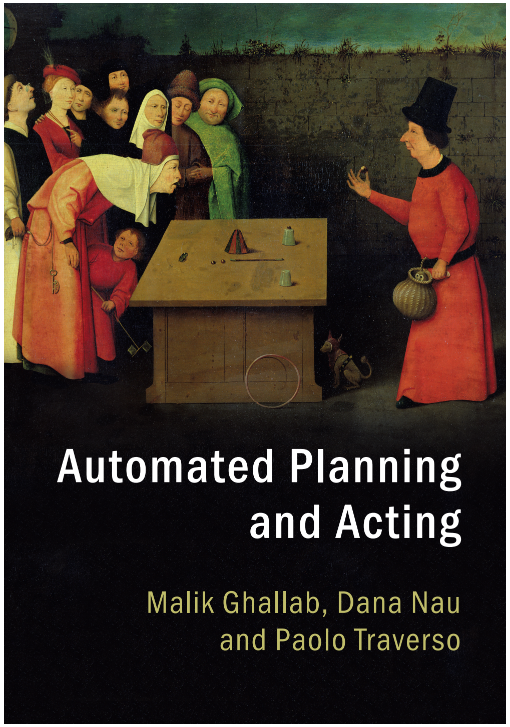
Last update: February 14, 2017



http://www.laas.fr/planning

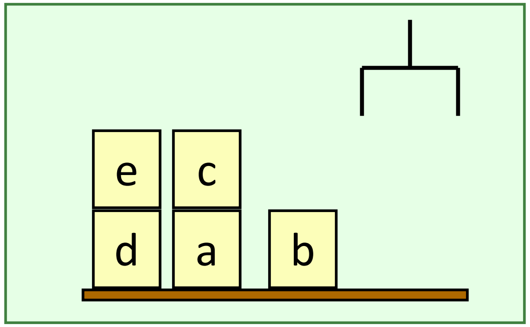
Section 2.7.8
Planning with Control Rules

Dana S. Nau
University of Maryland

Motivation

- Sometimes we can write highly efficient planning algorithms for a specific domain
  - Use special properties of the domain
- Example: the "blocks world"

clear(a)=F, clear(b)=T, clear(c)=T, clear(d)=F, clear(e)=T, holding=nil, loc(a)=table, loc(b)=table, loc(c)=a, loc(d)=table, loc(e)=d



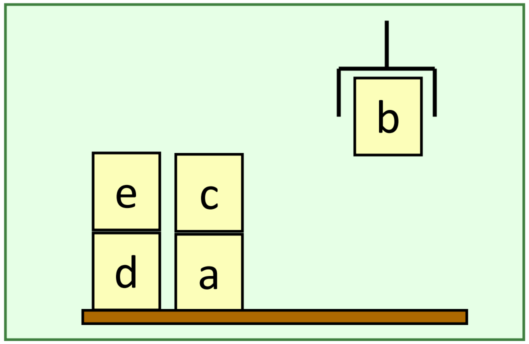
pickup(x)
  pre: loc(x)=table, clear(x)=T, holding=nil
  eff: loc(x)=crane, clear(x)=F, holding=x

putdown(x)
  pre: holding=x
  eff: holding=nil, loc(x)=table, clear(x)=T

stack(x,y)
  pre: holding=x, clear(y)=T
  eff: holding=nil, clear(y)=F, loc(x)=y, clear(x)=T

unstack(x,y)
  pre: loc(x)=y, clear(x)=T, holding=nil
  eff: loc(x)=crane, clear(x)=F, holding=x, clear(y)=T

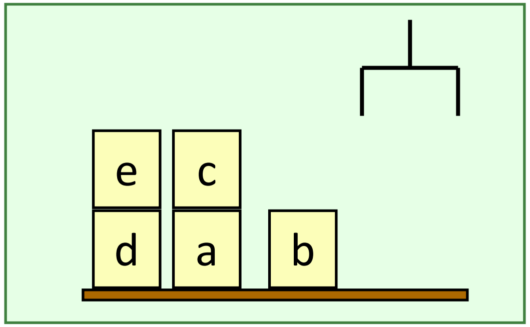
clear(a)=F, clear(b)=F, clear(c)=T, clear(d)=F, clear(e)=T, holding=b, loc(a)=table, loc(b)=crane, loc(c)=a, loc(d)=table, loc(e)=d



The Blocks World

- For block-stacking problems with n blocks, easy to get a solution of length $O(n)$
  - Move all blocks to the table, then build up stacks from the bottom
- With more domain knowledge, can do even better

clear(a)=F, clear(b)=T, clear(c)=T, clear(d)=F, clear(e)=T, holding=nil, loc(a)=table, loc(b)=table, loc(c)=a, loc(d)=table, loc(e)=d



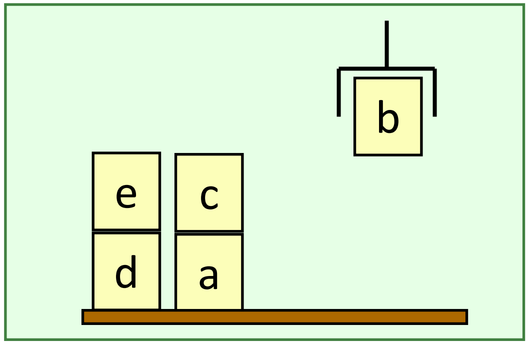
pickup(x)
  pre: loc(x)=table, clear(x)=T, holding=nil
  eff: loc(x)=crane, clear(x)=F, holding=x

putdown(x)
  pre: holding=x
  eff: holding=nil, loc(x)=table, clear(x)=T

stack(x,y)
  pre: holding=x, clear(y)=T
  eff: holding=nil, clear(y)=F, loc(x)=y, clear(x)=T

unstack(x,y)
  pre: loc(x)=y, clear(x)=T, holding=nil
  eff: loc(x)=crane, clear(x)=F, holding=x, clear(y)=T

clear(a)=F, clear(b)=F, clear(c)=T, clear(d)=F, clear(e)=T, holding=b, loc(a)=table, loc(b)=crane, loc(c)=a, loc(d)=table, loc(e)=d

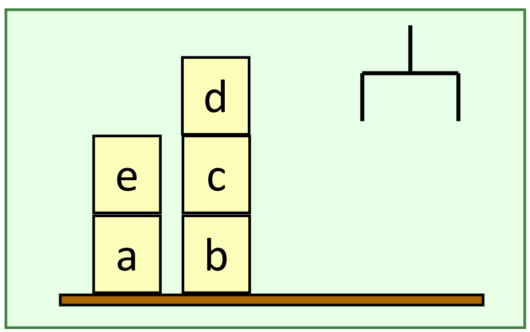


Block-Stacking Algorithm

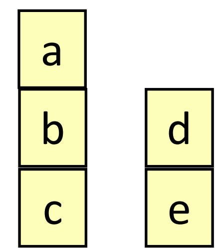
loop
   if ∃ a clear block c that needs moving
       & we can move c to a position d
          where c won't need to be moved
     then move c to d
   else if ∃ a clear block c that needs to be moved
       then move c to any clear pallet
   else if the goal is satisfied
       then return success
   else return failure
repeat

- c needs to be moved if
  - s contains loc(c)=d and g contains loc(c)=e, where $e \neq d$
  - s contains loc(c)=d and g contains loc(b)=d, where $b \neq c$
  - s contains loc(c)=d and d needs moving

s_0:



g:



⟨unstack(e,a), putdown(e), unstack(d,c), stack(d,e), unstack(c,b), putdown(c), pickup(b), stack(b,c), pickup(a), stack(a,b)⟩

Properties of the Algorithm

- Sound, complete, guaranteed to terminate on all block-stacking problems

- Runs in time $O(n^3)$
  - Can be modified (Slaney & Thiébaux) to run in time $O(n)$

- Often finds optimal (shortest) solutions
- But sometimes only near-optimal
  - For block-stacking problems, PLAN-LENGTH is NP-complete

- Some ways to implement it:
  - As a domain-specific algorithm
  - Using refinement methods (RAE and SeRPE, Chapter 3)
  - Using HTN planning (SHOP, PyHop, Section 2.7.7)
  - Using control rules

Planning with Control Rules

- Basic idea: given a state *s* and an action *a*, do domain-specific tests on $\gamma(s,a)$ to find cases where we won't want use *a*
    - *a* doesn't lead to a solution
    - *a* is *dominated* (there's a better solution along some other path)
    - *a* doesn't lead to a solution that's acceptable according to domain-specific criteria
  - ➢ In such cases, *prune s*
- Write logical formulas giving conditions that states must satisfy
  - ➢ Prune states that don't satisfy the formulas

Quick Review of First Order Logic

- First Order Logic (FOL) syntax:
  - atomic formulas (or *atoms*)
    - predicate symbol with arguments, e.g., clear(c)
  - logical connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$), quantifiers ($\forall$, $\exists$), punctuation
    - e.g., (loc(r1)=d1 $\wedge$ $\forall c$ clear(c)) $\Rightarrow$ $\neg\exists c$ loc(c)=r1
- FOL with equality
  - '=' is a binary predicate symbol, e.g., loc(r1)=d1
- First Order Theory $\mathcal{T}$
  - "Logical" axioms, inference rules – encode logical reasoning in general
  - Additional "nonlogical" axioms – talk about a particular domain
  - Theorems: produced by applying the axioms and rules of inference
- *Model*: a set of objects, functions, relations that the symbols refer to
  - For our purposes, a model is a state of the world s
  - In order for s to be a model, all theorems of $\mathcal{T}$ must be true in s
  - $s \vDash$ loc(r1)=d1 "s satisfies loc(r1)=d1" or "s entails loc(r1)=d1"
    - r1 is at d1 in the state s

Linear Temporal Logic

- *Modal logic*: FOL plus *modal operators*
  to express concepts that would be difficult to express within FOL

- Linear Temporal Logic (LTL):
  - Purpose: to express a limited notion of time
    - Infinite sequence ⟨0, 1, 2, …⟩ of time instants
    - Infinite sequence $M = \langle s_0, s_1, \ldots \rangle$ of states of the world
  - Modal operators to refer to states in M:

    X f "next f" - f is true in the next state, e.g., X loc(a)=b

    F f "future f" - f either is true now or in some future state

    G f "globally f" - f is true now and in all future states

    f_1 U f_2 "f_1 until f_2" - f_2 is true now or in a future state, and f_1 is true until then
  - Propositional constant symbols true and false

Linear Temporal Logic (continued)

- Quantifiers cause problems with computability
  - Suppose $f(x)$ is true for infinitely many values of x
  - Problem evaluating truth of $\forall x\ f(x)$ and $\exists x\ f(x)$

- Bounded quantifiers
  - Let $g(x)$ be such that $\{x \mid g(x) \text{ is true}\}$ is finite and easily computed

    $\forall[x\!: g(x)]\, f(x)$
    - means $\forall x\ (g(x) \Rightarrow f(x))$
    - expands into $f(x_1) \wedge f(x_2) \wedge \ldots \wedge f(x_n)$

    $\exists[x\!: g(x)]\, f(x)$
    - means $\exists x\ (g(x) \wedge f(x))$
    - expands into $f(x_1) \vee f(x_2) \vee \ldots \vee f(x_n)$

State-Variable Notation in LTL Formulas

- We can use state-variable assignments directly as atoms
  - clear(c)=T $\wedge$ X loc(a)=c

- Simplify the notation
  - Earlier we defined clear(x) to be Boolean, i.e., Range(clear(x)) = {T,F}
  - Can replace it with a logical proposition
    - Instead of writing clear(x)=T, write clear(x)
    - Instead of writing clear(x)=F, write $\neg$clear(x)

  - clear(c) $\wedge$ X loc(a)=c

Examples

- Suppose $M = \langle s_0, s_1, \ldots \rangle$

- All of the following are equivalent:
    - All mean a is on b in state s_2
  - $(M,s_0) \vDash \mathrm{XX}$ loc(a)=b
  - $M \vDash \mathrm{XX}$ loc(a)=b   omit the state, it defaults to s_0
  - $(M,s_2) \vDash$ loc(a)=b
  - $s_2 \vDash$ loc(a)=b

- $M \vDash \mathrm{G}$ holding ≠ c
  - in every state in *M,* we aren't holding c

- $M \vDash \mathrm{G}$ (clear(b) $\Rightarrow$ (clear(b) U loc(a)=b))
  - whenever we enter a state in which b is clear,
    b remains clear until a is on b

Models for Planning with LTL

- A model is a pair $\mathcal{M} = (M, s_i)$
  - $M = \langle s_0, s_1, \ldots \rangle$ is a sequence of states
  - s_i is the i'th state in M,
- For planning, we also have a goal $g = \{g_1, \ldots, g_n\}$
  - To reason about it, add a modal operator called "Goal"
    - Not part of ordinary LTL, but I'll call it LTL anyway
  - In an LTL formula, use "Goal(g_i)" to refer to part of g
    - $((M,s_i), g) \vDash \text{Goal}(g_i)$ iff $g \vDash g_i$
- Planning problem:
  - Initial state s_0, a goal g, control formula f
  - Find a plan $\pi = \langle a_1, \ldots, a_n \rangle$ that generates a sequence of states $M = \langle s_0, s_1, \ldots s_n \rangle$ such that $M \vDash f$ and $s_n \vDash g$
    - That's not quite correct
    - Do you know why?

Models for Planning with LTL

- M needs to be an infinite sequence
- Kluge: assume that the final state repeats infinitely after the plan ends

- Planning problem:
  - Initial state s_0, a goal g, control formula f
  - Find a plan $\pi = \langle a_1, \ldots, a_n \rangle$ that generates a sequence of states $M = \langle s_0, s_1, \ldots, s_n, s_n, s_n, \ldots \rangle$ such that $M \vDash f$ and $s_n \vDash g$

TLPlan

- Nondeterministic forward search
  - s = current state, f = control formula, g = goal
- If s satisfies g then we're done
- Otherwise, think about what kind of plan we need
  - It must generate a sequence of states $M = \langle s, s^+, s^{++}, \ldots \rangle$ that satisfies f
- Compute a formula f^+ such that
  - $(M,s) \vDash f$ iff $(M,s^+) \vDash f^+$
- If f^+ = false, then fail
  - No matter what M and s^+ are, they can't satisfy f^+
- Fail if no applicable actions
- Otherwise, nondeterministically choose one, compute s^+, and call TLPlan with s^+ and f^+

```
TLPlan (s, f, g)
    if s satisfies g then return ⟨ ⟩
    f⁺ ← Progress (f, s)
    if f⁺ = false then return failure
    A ← {actions applicable to s}
    if A is empty then return failure
    nondeterministically choose a ∈ A
    s⁺ = γ(s,a)
    π⁺ ← TLPlan (s⁺, f⁺, g)
    if π⁺ ≠ failure then return a.π⁺
    return failure
```

Progression

Procedure Progress(f,s)

Compute the formula f^+ that M^+ must satisfy

Case:

1. f contains no temporal ops : $f^+ \leftarrow$ true if $s \models f$, false otherwise
2. $f = f_1 \wedge f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge$ Progress(f_2, s)
3. $f = f_1 \vee f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\vee$ Progress(f_2, s)
4. $f = \neg f_1$: $f^+ \leftarrow \neg$Progress(f_1, s)
5. $f = \mathrm{X}\, f_1$: $f^+ \leftarrow f_1$
6. $f = \mathrm{F}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\vee f$
7. $f = \mathrm{G}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge f$
8. $f = f_1 \,\mathrm{U}\, f_2$: $f^+ \leftarrow$ Progress(f_2, s) $\vee$ (Progress(f_1, s) $\wedge f$)
9. $f = \forall[x{:}g(x)]\; h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\wedge \ldots \wedge$ Progress($h(x_n)$, s)
10. $f = \exists\,[x{:}g(x)]\; h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\vee \ldots \vee$ Progress($h(x_n)$, s)

simplify f^+ and return it

false $\wedge\, h$ = false,
true $\wedge\, h = h$,
$\neg$false = true,
etc.

Progressing ordinary formulas

Procedure Progress(f,s)

Case:

1.	f contains no temporal ops	:	$f^+ \leftarrow$ true if $s \models f$, false otherwise
2.	$f = f_1 \wedge f_2$	:	$f^+ \leftarrow$ Progress(f_1, s) $\wedge$ Progress(f_2, s)
3.	$f = f_1 \vee f_2$	:	$f^+ \leftarrow$ Progress(f_1, s) $\vee$ Progress(f_2, s)
4.	$f = \neg f_1$	:	$f^+ \leftarrow \neg$Progress(f_1, s)
5.	$f = \mathrm{X}\, f_1$	:	$f^+ \leftarrow f_1$
6.	$f = \mathrm{F}\, f_1$	:	$f^+ \leftarrow$ Progress(f_1, s) $\vee f$
7.	$f = \mathrm{G}\, f_1$	:	$f^+ \leftarrow$ Progress(f_1, s) $\wedge f$
8.	$f = f_1 \,\mathrm{U}\, f_2$	:	$f^+ \leftarrow$ Progress(f_2, s) $\vee$ (Progress(f_1, s) $\wedge f$)
9.	$f = \forall[x{:}g(x)]\ h(x)$	:	$f^+ \leftarrow$ Progress($h(x_1)$, s) $\wedge \ldots \wedge$ Progress($h(x_n)$, s)
10.	$f = \exists\,[x{:}g(x)]\ h(x)$	:	$f^+ \leftarrow$ Progress($h(x_1)$, s) $\vee \ldots \vee$ Progress($h(x_n)$, s)

simplify f^+ and return it

- f = loc(a)=b
  - if a is currently on b, then true (every possible M^+ is OK)
  - otherwise false (there is no M^+ that's OK)

Progressing X

Procedure Progress(f,s)

Case:

1. f contains no temporal ops : $f^+ \leftarrow$ true if $s \vDash f$, false otherwise
2. $f = f_1 \wedge f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge$ Progress(f_2, s)
3. $f = f_1 \vee f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\vee$ Progress(f_2, s)
4. $f = \neg f_1$: $f^+ \leftarrow \neg$Progress(f_1, s)
5. $f = \mathrm{X}\, f_1$: $f^+ \leftarrow f_1$
6. $f = \mathrm{F}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\vee f$
7. $f = \mathrm{G}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge f$
8. $f = f_1 \,\mathrm{U}\, f_2$: $f^+ \leftarrow$ Progress(f_2, s) $\vee$ (Progress(f_1, s) $\wedge f$)
9. $f = \forall[x{:}g(x)]\ h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\wedge \ldots \wedge$ Progress($h(x_n)$, s)
10. $f = \exists\,[x{:}g(x)]\ h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\vee \ldots \vee$ Progress($h(x_n)$, s)

simplify f^+ and return it

- $f = \mathrm{X}$ loc(a)=b
  - in the next state, a must be on b
  - $f^+ =$ loc(a)=b

- $f = \mathrm{XX}$ loc(a)=b
  - two states from now, a must be on b
  - $f^+ = \mathrm{X}$ loc(a)=b

Progressing $\wedge$

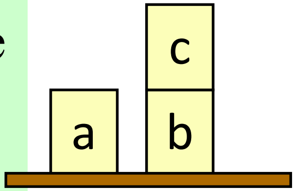
Procedure Progress(f,s)

Case:

1. f contains no temporal ops : $f^+ \leftarrow$ true if $s \vDash f$, false otherwise
2. $f = f_1 \wedge f_2$: $f^+ \leftarrow \text{Progress}(f_1, s) \wedge \text{Progress}(f_2, s)$
3. $f = f_1 \vee f_2$: $f^+ \leftarrow \text{Progress}(f_1, s) \vee \text{Progress}(f_2, s)$
4. $f = \neg f_1$: $f^+ \leftarrow \neg\text{Progress}(f_1, s)$
5. $f = \text{X}\, f_1$: $f^+ \leftarrow f_1$
6. $f = \text{F}\, f_1$: $f^+ \leftarrow \text{Progress}(f_1, s) \vee f$
7. $f = \text{G}\, f_1$: $f^+ \leftarrow \text{Progress}(f_1, s) \wedge f$
8. $f = f_1 \,\text{U}\, f_2$: $f^+ \leftarrow \text{Progress}(f_2, s) \vee (\text{Progress}(f_1, s) \wedge f)$
9. $f = \forall[x{:}g(x)]\ h(x)$: $f^+ \leftarrow \text{Progress}(h(x_1), s) \wedge \ldots \wedge \text{Progress}(h(x_n), s)$
10. $f = \exists\,[x{:}g(x)]\ h(x)$: $f^+ \leftarrow \text{Progress}(h(x_1), s) \vee \ldots \vee \text{Progress}(h(x_n), s)$

simplify f^+ and return it

- f = clear(c)=T $\wedge$ X loc(a)=c
  - c must be clear now, and a must be on c in the next state
- f^+ = Progress(clear(c)=T, s) $\wedge$ Progress(X loc(a)=c, s)
  = true $\wedge$ loc(a)=c
  = loc(a)=c



Progressing $\wedge$

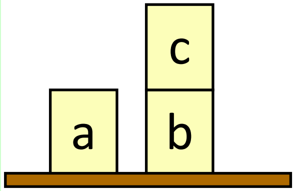
Procedure Progress(f,s)

Case:

1. f contains no temporal ops : $f^+ \leftarrow$ true if $s \vDash f$, false otherwise
2. $f = f_1 \wedge f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge$ Progress(f_2, s)
3. $f = f_1 \vee f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\vee$ Progress(f_2, s)
4. $f = \neg f_1$: $f^+ \leftarrow \neg$Progress(f_1, s)
5. $f = \mathrm{X}\, f_1$: $f^+ \leftarrow f_1$
6. $f = \mathrm{F}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\vee f$
7. $f = \mathrm{G}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge f$
8. $f = f_1\ \mathrm{U}\ f_2$: $f^+ \leftarrow$ Progress(f_2, s) $\vee$ (Progress(f_1, s) $\wedge f$)
9. $f = \forall[x{:}g(x)]\ h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\wedge \ldots \wedge$ Progress($h(x_n)$, s)
10. $f = \exists\,[x{:}g(x)]\ h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\vee \ldots \vee$ Progress($h(x_n)$, s)

simplify f^+ and return it

- $f = \mathrm{G}$ loc(a)=c
  - a must be on c now, and must stay there forever
- $f^+ =$ Progress(loc(a)=c, s) $\wedge\ f$
  $=$ false $\wedge\ \mathrm{G}$ loc(a)=c
  $=$ false



Progressing $\wedge$

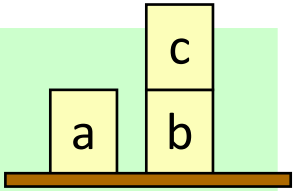
Procedure Progress(f,s)

Case:

1. f contains no temporal ops : $f^+ \leftarrow$ true if $s \vDash f$, false otherwise
2. $f = f_1 \wedge f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge$ Progress(f_2, s)
3. $f = f_1 \vee f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\vee$ Progress(f_2, s)
4. $f = \neg f_1$: $f^+ \leftarrow \neg$Progress(f_1, s)
5. $f = \mathrm{X}\, f_1$: $f^+ \leftarrow f_1$
6. $f = \mathrm{F}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\vee f$
7. $f = \mathrm{G}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge f$
8. $f = f_1 \,\mathrm{U}\, f_2$: $f^+ \leftarrow$ Progress(f_2, s) $\vee$ (Progress(f_1, s) $\wedge f$)
9. $f = \forall[x{:}g(x)]\; h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\wedge \ldots \wedge$ Progress($h(x_n)$, s)
10. $f = \exists\,[x{:}g(x)]\; h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\vee \ldots \vee$ Progress($h(x_n)$, s)

simplify f^+ and return it

- $f =$ loc(a)=b U clear(c)=T
  - c must be clear, **or** a must be on b and stay there until c is clear



- $f^+ =$ Progress(clear(c)=T, s) $\vee$ [Progress(loc(a)=b, s) $\wedge$ f]
  $=$ true $\vee$ [false $\wedge$ (loc(a)=b) U clear(c)=T)]
  $=$ true

Progressing ∀

Procedure Progress(f,s)

Case:

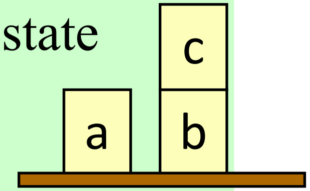
1. f contains no temporal ops : $f^+ \leftarrow$ true if $s \vDash f$, false otherwise
2. $f = f_1 \wedge f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge$ Progress(f_2, s)
3. $f = f_1 \vee f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\vee$ Progress(f_2, s)
4. $f = \neg f_1$: $f^+ \leftarrow \neg$Progress(f_1, s)
5. $f = \mathrm{X}\, f_1$: $f^+ \leftarrow f_1$
6. $f = \mathrm{F}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\vee f$
7. $f = \mathrm{G}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge f$
8. $f = f_1\ \mathrm{U}\ f_2$: $f^+ \leftarrow$ Progress(f_2, s) $\vee$ (Progress(f_1, s) $\wedge f$)
9. $f = \forall[x{:}g(x)]\ h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\wedge \ldots \wedge$ Progress($h(x_n)$, s)
10. $f = \exists\,[x{:}g(x)]\ h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\vee \ldots \vee$ Progress($h(x_n)$, s)

x_i is the i'th element of $\{x \mid s \vDash g(x)\}$

simplify f^+ and return it

- $f = \forall[x$: clear(x)=T] X loc(x)=table
  - every currently-clear block must be on the table in the next state
- f^+ = Progress(X loc(a)=table, s) $\wedge$ Progress(X loc(c)=table, s)

  = loc(a)=table $\wedge$ loc(c)=table



Progressing ∃

Procedure Progress(f, s)

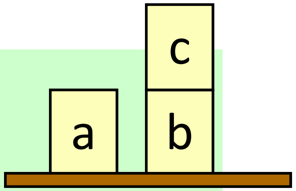
Case:

1. f contains no temporal ops : $f^+ \leftarrow$ true if $s \vDash f$, false otherwise
2. $f = f_1 \wedge f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge$ Progress(f_2, s)
3. $f = f_1 \vee f_2$: $f^+ \leftarrow$ Progress(f_1, s) $\vee$ Progress(f_2, s)
4. $f = \neg f_1$: $f^+ \leftarrow \neg$Progress(f_1, s)
5. $f = \mathrm{X}\, f_1$: $f^+ \leftarrow f_1$
6. $f = \mathrm{F}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\vee f$
7. $f = \mathrm{G}\, f_1$: $f^+ \leftarrow$ Progress(f_1, s) $\wedge f$
8. $f = f_1\, \mathrm{U}\, f_2$: $f^+ \leftarrow$ Progress(f_2, s) $\vee$ (Progress(f_1, s) $\wedge f$)
9. $f = \forall[x{:}g(x)]\; h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\wedge \ldots \wedge$ Progress($h(x_n)$, s)
10. $f = \exists\,[x{:}g(x)]\; h(x)$: $f^+ \leftarrow$ Progress($h(x_1)$, s) $\vee \ldots \vee$ Progress($h(x_n)$, s)

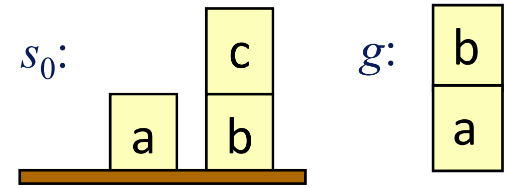
x_i is the i'th element of $\{x \mid s \vDash g(x)\}$

simplify f^+ and return it

- $f = \exists[x$: clear(x)=T] X loc(x)=table
  - $\{x \mid$ clear(x)=T$\}$ = {a, c}
- f^+ = Progress(X loc(a)=table, s) $\vee$ Progress(X loc(c)=table, s)
  = loc(a)=table $\vee$ loc(c)=table



Example Planning Problem



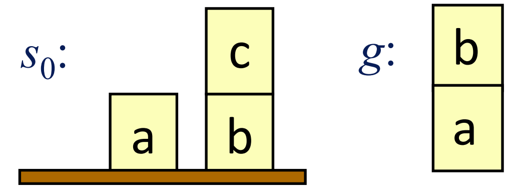
- s = {clear(a)=T, clear(b)=F, clear(c)=T, holding=nil, loc(a)=table, loc(b)=table, loc(c)=b}
- g = {loc(b)=a}
- f = G $\forall[x$: clear(x)] (loc(x)$\neq$table $\vee$ $\exists[y$: Goal(loc(x)=y)] $\vee$ X holding$\neq x$)
  - never pick up a clear block from the table unless it needs to be elsewhere

Run TLPlan using depth-first search

- s doesn't satisfy g
- Compute f^+ (see next page)
  - f^+ = holding≠a $\wedge f$
- A = {pickup(a), unstack(c,b)}
- Try using pickup(a)
  - $s^+ = \gamma(s$, pickup(a))
  - Call TLPlan (s^+, f^+, g)
    - Progress(f^+, s^+) = false
  - Recursive call returns failure
- Try unstack(c,b) …

TLPlan (s, f, g)
    if s satisfies g then return $\langle\,\rangle$
    $f^+ \leftarrow$ Progress (f, s)
    if f^+ = false then return failure
    $A \leftarrow$ {actions applicable to s}
    if A is empty then return failure
    nondeterministically choose $a \in A$
    $s^+ = \gamma(s,a)$
    $\pi^+ \leftarrow$ TLPlan (s^+, f^+, g)
    if $\pi^+ \neq$ failure then return $a.\pi^+$
    return failure

Computing f^+



- $s = \{$loc(a)=table, loc(b)=table, clear(a), clear(c), loc(c)=b$\}$
- $g = \{$loc(b)=a$\}$
- $f = \mathrm{G}\ \underbrace{\forall[x\text{: clear}(x)]\ \overbrace{(\text{loc}(x){\neq}\text{table} \vee \exists[y\text{: Goal}(\text{loc}(x){=}y)] \vee \mathrm{X}\ \text{holding}{\neq}x)}^{h(x)}}_{f_1}$

- $f^+ = \text{Progress}(\mathrm{G}\,f_1, s) = \text{Progress}(f_1, s) \wedge f$
  $= \text{Progress}(\forall[x\text{: clear}(x)]\ h(x)), s) \wedge f$
  $= \text{Progress}(h(\text{a}) \wedge h(\text{c})), s) \wedge f$
  $= \text{Progress}(h(\text{a})), s) \wedge \text{Progress}(h(\text{c})), s) \wedge f$

  - $\text{Progress}(h(\text{a}),s) = \text{Progress}(\text{loc(a)}{\neq}\text{table} \vee \exists[y\text{: Goal}(\text{loc(a)}{=}y)] \vee \mathrm{X}\ \text{holding}{\neq}\text{a}), s)$
    $= \text{false} \vee \text{false} \vee \text{holding}{\neq}\text{a}$
    $= \text{holding}{\neq}\text{a}$
  - $\text{Progress}(h(\text{c}),s) = \text{Progress}(\text{loc(c)}{\neq}\text{table} \vee \exists[y\text{: Goal}(\text{loc(c)}{=}y)] \vee \mathrm{X}\ \text{holding}{\neq}\text{c}), s)$
    $= \text{true} \vee \text{false} \vee \text{holding}{\neq}\text{c}$
    $= \text{true}$
- $f^+ = \text{holding}{\neq}\text{a} \wedge \text{true} \wedge f = \text{holding}{\neq}\text{a} \wedge f$

Block-Stacking Problems

- Want to define a formula final(x) that means
  - x is at the top of a stack and we're finished moving it
  - Neither x nor the blocks below x will ever need to be moved
- Axioms to support this:
  - final(x) $\Leftrightarrow$ clear(x) $\wedge$ $\neg$Goal(holding=x) $\wedge$ finalbelow(x)
  - finalbelow(x) $\Leftrightarrow$

    (loc(x)=table $\wedge$ $\forall$[y: Goal(loc(x)=y] y=table)

    $\vee$ $\exists$[y: loc(x)=y] [

    $\neg$Goal(loc(x)=table) $\wedge$ $\neg$Goal(holding=y) $\wedge$ $\neg$Goal(clear(y))

    $\wedge$ $\forall$[z : Goal(loc(x)=z)] (z=y) $\wedge$ $\forall$[z: Goal(loc(z)=y)] (z=x)

    $\wedge$ finalbelow(y)]
  - nonfinal(x) $\Leftrightarrow$ clear(x) $\wedge$ $\neg$final(x)

Control Rules

Try TLPlan with three different control formulas:

(1) If x is final, only put a block y onto x if it will make y final:

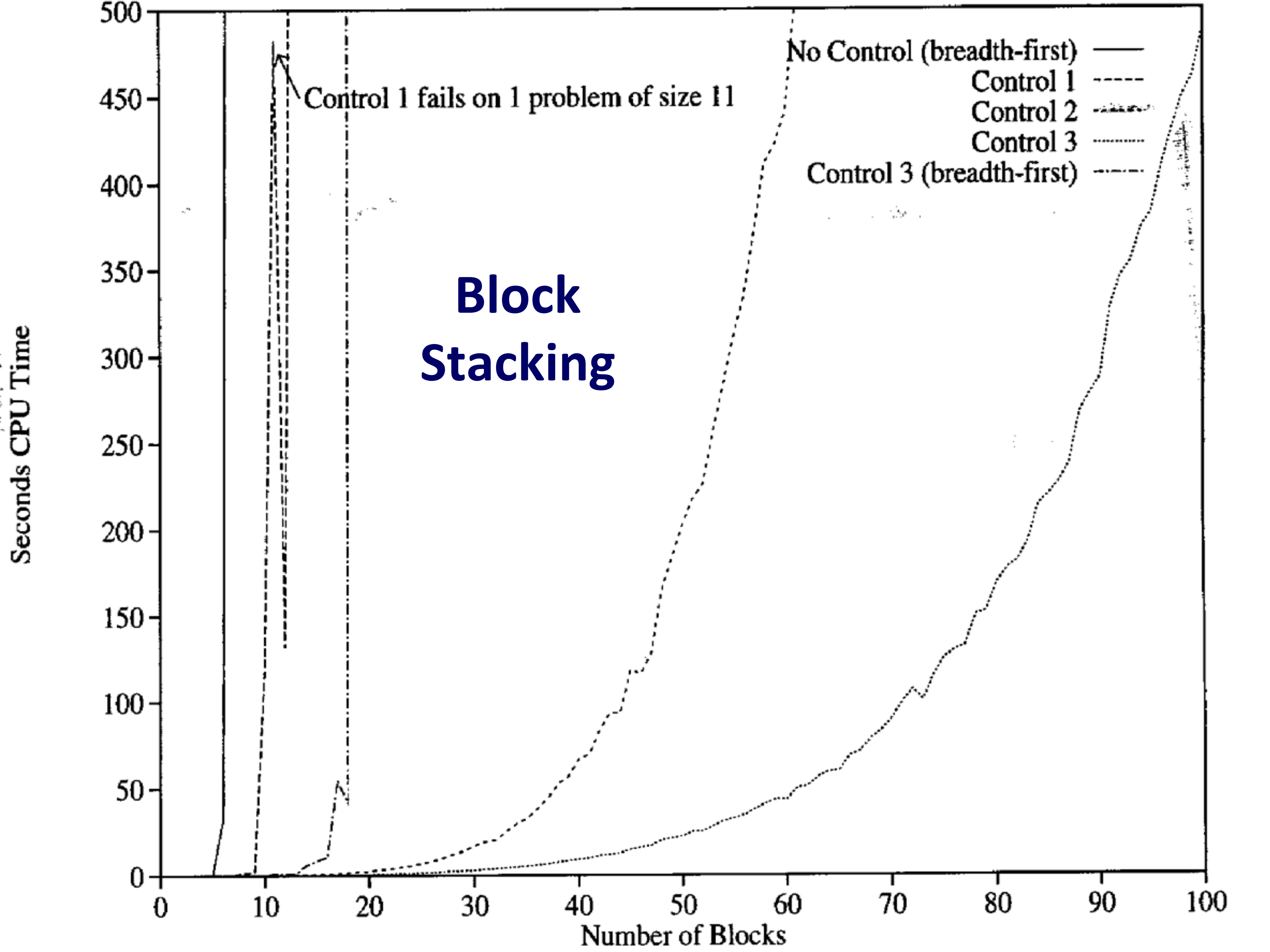
- $G \, \forall [x\text{: clear}(x)] \, (\text{final}(x) \Rightarrow X \, [\text{clear}(x) \vee \exists [y\text{: loc}(y)=x] \, \text{final}(y)])$

(2) Like (1), but also says that if a block isn't final, don't put anything onto it:

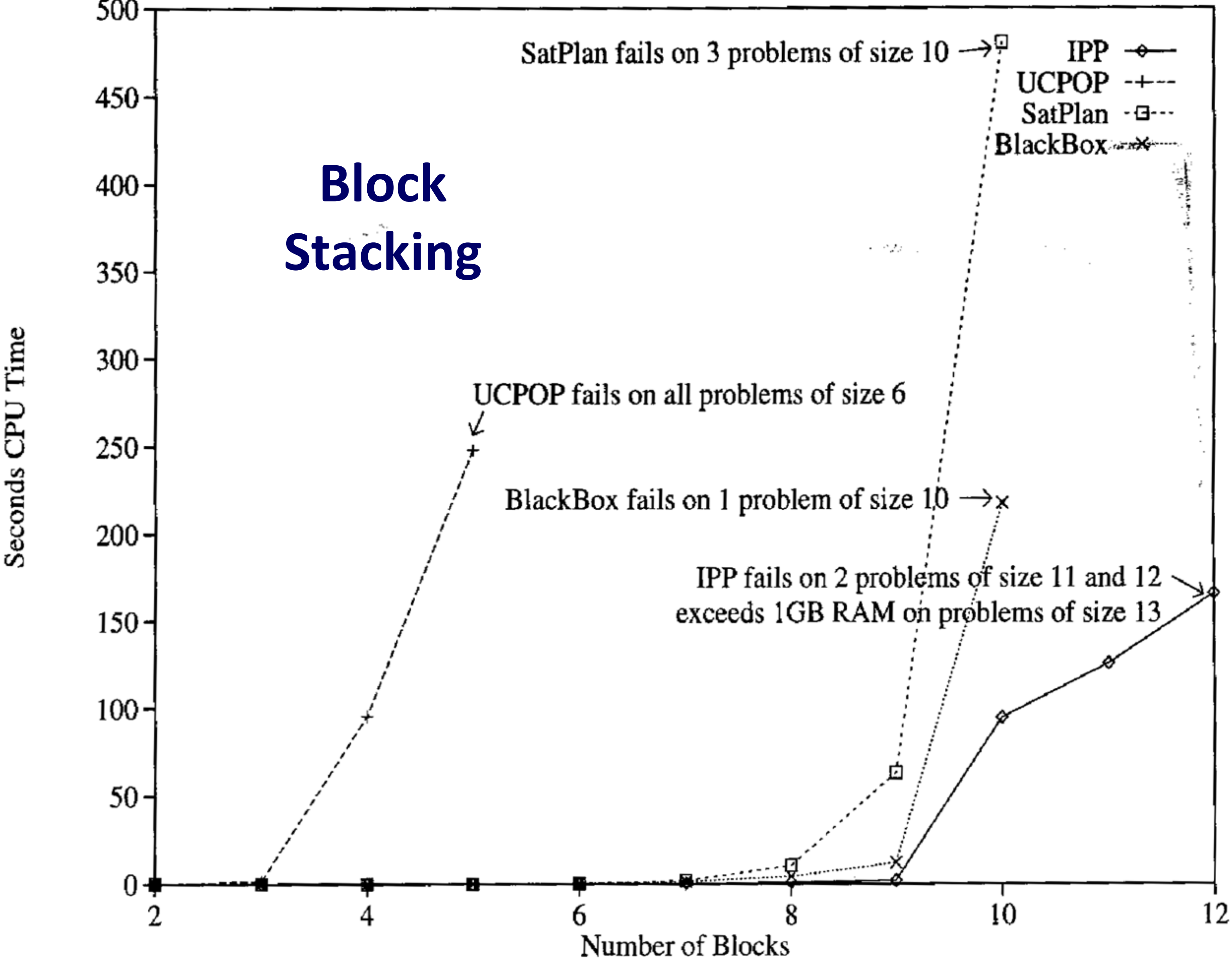
- $G \, \forall [x\text{: clear}(x)] \, [$
  $(\text{final}(x) \Rightarrow X \, [\text{clear}(x) \vee \exists [y\text{: loc}(y)=x] \, \text{final}(y)])$
  $\wedge \, (\text{nonfinal}(x) \Rightarrow X \neg \exists [y\text{: loc}(y)=x])]$

(3) Like (2), but also says not to pick up a nonfinal block from the table unless you can put it where it will be final:

- $G \, \forall [x\text{: clear}(x)] \, [$
  $(\text{final}(x) \Rightarrow X \, [\text{clear}(x) \vee \exists [y\text{: loc}(y)=x] \, \text{final}(y)])$
  $\wedge \, (\text{nonfinal}(x) \Rightarrow X \neg \exists [y\text{: loc}(y)=x])$
  $\wedge \, (\text{ontable}(x) \wedge \exists [y\text{: Goal}(\text{loc}(x)=y)] \, [\neg \text{final}(y) \Rightarrow X \neg \text{holding}(x)])]$



Block Stacking
Seconds CPU Time
Number of Blocks
No Control (breadth-first)
Control 1
Control 2
Control 3
Control 3 (breadth-first)
Control 1 fails on 1 problem of size 11
0
50
100
150
200
250
300
350
400
450
500
0
10
20
30
40
50
60
70
80
90
100



Block Stacking
Seconds CPU Time
Number of Blocks
IPP
UCPOP
SatPlan
BlackBox
SatPlan fails on 3 problems of size 10
UCPOP fails on all problems of size 6
BlackBox fails on 1 problem of size 10
IPP fails on 2 problems of size 11 and 12
exceeds 1GB RAM on problems of size 13
0
50
100
150
200
250
300
350
400
450
500
2
4
6
8
10
12