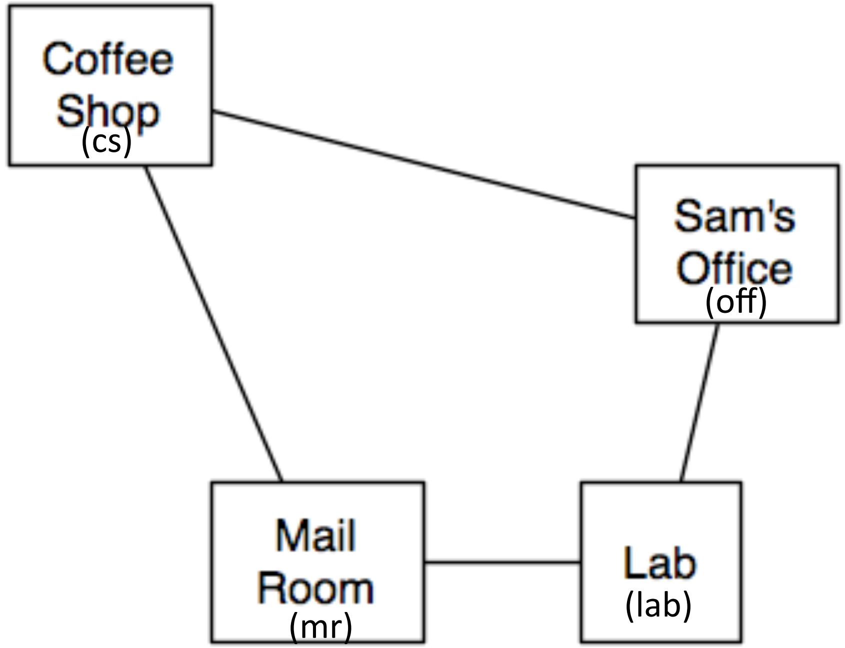
CS 4260 and CS 5260
Vanderbilt University

Lecture on Propositional Planning

This lecture assumes that you have
- Read section 4.1, section 5.1, section 2.4, and Chapter 6 through 6.3 of ArtInt

ArtInt: Poole and Mackworth, Artificial Intelligence 2E
at http://artint.info/2e/html/ArtInt2e.html

Example 6.1 Consider a delivery robot world with mail and coffee to deliver. Assume a simplified domain with four locations as shown in Figure 6.1



From ArtInt

Features to describe states

RLoc
– Rob's location
RHC
– Rob has coffee
SWC
– Sam wants coffee
MW
– Mail is waiting
RHM
– Rob has mail

Actions

mc
– move clockwise
mcc
– move counterclockwise
puc
– pickup coffee
dc
– deliver coffee
pum
– pickup mail
dm
– deliver mail

Explicit State-Space Representation

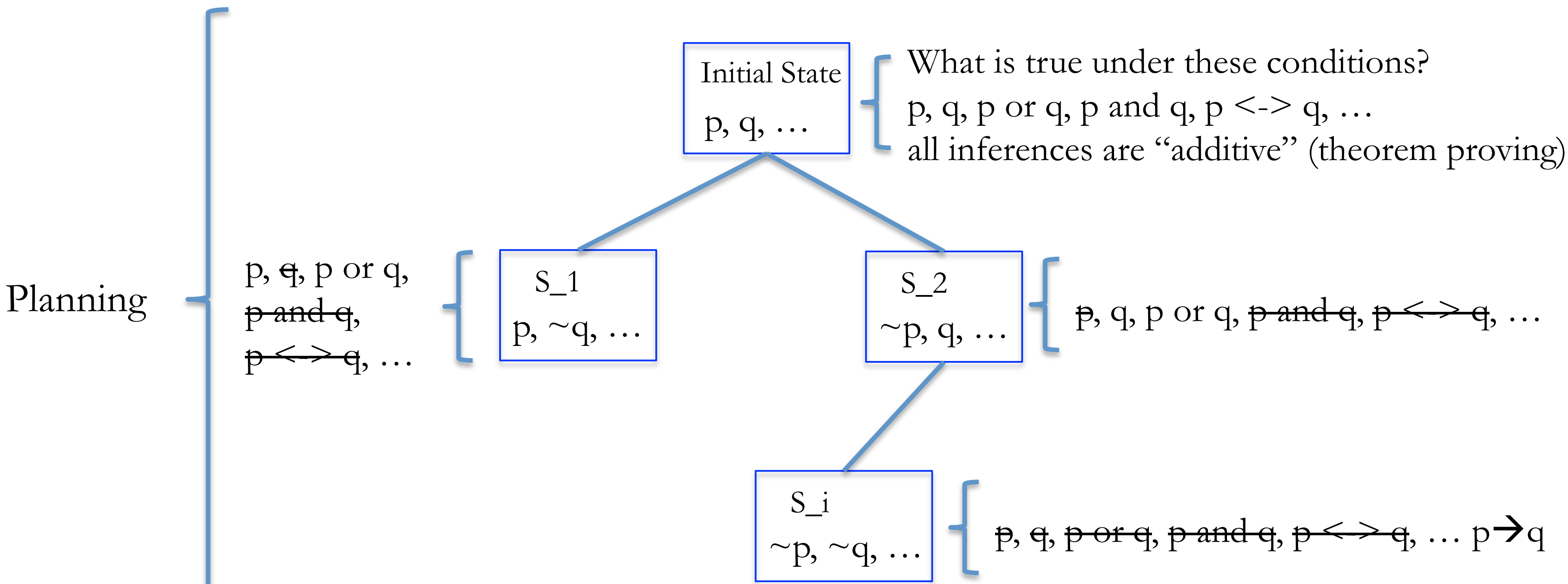
State	Action	Resulting State
⟨*lab*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩	*mc*	⟨*mr*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩
⟨*lab*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩	*mcc*	⟨*off*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩
⟨*off*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩	*dm*	⟨*off*, ¬*rhc*,*swc*, ¬*mw*, ¬*rhm*⟩
⟨*off*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩	*mcc*	⟨*cs*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩
⟨*off*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩	*mc*	⟨*lab*, ¬*rhc*,*swc*, ¬*mw*,*rhm*⟩
...	...	...

An important aside

An AI uses search (e.g., DFD, BFS, HDFS, GBFS, IDDFS, A*, IDA*)

- to solve (constraint) problems (chapter 4),
- to prove theorems (chapter 5), and
- to plan actions (chapter 6) – today's focus

Much of chapters 4 and 5 are about reasoning in "static worlds", in which a knowledge that models the world does not change (at least while reasoning is proceeding). We will talk more about chapter 5's concepts of knowledge bases, interpretations, and models week after next, but you will receive some quiz feedback on it this week.



An explicit state-space representation can be cumbersome

- Too many states
- Fragile
- No pattern explicit

Adapted from ArtInt

Features to describe states

RLoc
– Rob's location (4-valued)

RHC
– Rob has coffee (binary)

SWC
– Sam wants coffee (binary)

MW
– Mail is waiting (binary)

RHM
– Rob has mail (binary)

Actions

mc
– move clockwise

mcc
– move counterclockwise

puc
– pickup coffee

dc
– deliver coffee

pum
– pickup mail

dm
– deliver mail

State	Action	Resulting State
< lab, rhc, swc, mw, rhm>	mc	< mr, rhc, swc, mw, rhm>
< lab, rhc, swc, mw, ~rhm>	mc	< mr, rhc, swc, mw, ~rhm>
< lab, rhc, swc, ~mw, rhm>	mc	< mr, rhc, swc, ~mw, rhm>
< lab, rhc, swc, ~mw, ~rhm>	mc	< mr, rhc, swc, ~mw, ~rhm>
< lab, rhc, ~swc, mw, rhm>	mc	< mr, rhc, ~swc, mw, rhm>
< lab, rhc, ~swc, mw, ~rhm>	mc	< mr, rhc, ~swc, mw, ~rhm>
< lab, rhc, ~swc, ~mw, rhm>	mc	< mr, rhc, ~swc, ~mw, rhm>
< lab, rhc, ~swc, ~mw, ~rhm>	mc	< mr, rhc, ~swc, ~mw, ~rhm>
< lab, ~rhc, swc, mw, rhm>	mc	< mr, ~rhc, swc, mw, rhm>
…		
< lab, ~rhc, ~swc, ~mw, ~rhm>	mc	< mr, ~rhc, ~swc, ~mw, ~rhm>

State	Action	Resulting State
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨mr, ¬rhc,swc, ¬mw,rhm⟩*
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨off, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*dm*	*⟨off, ¬rhc,swc, ¬mw, ¬rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨cs, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨lab, ¬rhc,swc, ¬mw,rhm⟩*
...	...	...

An aside: unstructured representations can be structured through learning

Adapted from ArtInt

Features to describe states

RLoc
- Rob's location (4-valued)

RHC
- Rob has coffee (binary)

SWC
- Sam wants coffee (binary)

MW
- Mail is waiting (binary)

RHM
- Rob has mail (binary)

Actions

mc
- move clockwise

mcc
- move counterclockwise

puc
- pickup coffee

dc
- deliver coffee

pum
- pickup mail

dm
- deliver mail

State	Action	Resulting State
< lab, rhc, swc, mw, rhm>	mc	< mr, rhc, swc, mw, rhm>
< lab, rhc, swc, mw, ~rhm>	mc	< mr, rhc, swc, mw, ~rhm>
< lab, rhc, swc, ~mw, rhm>	mc	< mr, rhc, swc, ~mw, rhm>
< lab, rhc, swc, ~mw, ~rhm>	mc	< mr, rhc, swc, ~mw, ~rhm>
< lab, rhc, ~swc, mw, rhm>	mc	< mr, rhc, ~swc, mw, rhm>
< lab, rhc, ~swc, mw, ~rhm>	mc	< mr, rhc, ~swc, mw, ~rhm>
< lab, rhc, ~swc, ~mw, rhm>	mc	< mr, rhc, ~swc, ~mw, rhm>
< lab, rhc, ~swc, ~mw, ~rhm>	mc	< mr, rhc, ~swc, ~mw, ~rhm>
< lab, ~rhc, swc, mw, rhm>	mc	< mr, ~rhc, swc, mw, rhm>
…		
< lab, ~rhc, ~swc, ~mw, ~rhm>	mc	< mr, ~rhc, ~swc, ~mw, ~rhm>
<lab, ?V1, ?V2, ?V3, ?V4>	**mc**	**<mr, ?V1, ?V2, ?V3, ?V4>**

State	Action	Resulting State
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨mr, ¬rhc,swc, ¬mw,rhm⟩*
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨off, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*dm*	*⟨off, ¬rhc,swc, ¬mw, ¬rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨cs, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨lab, ¬rhc,swc, ¬mw,rhm⟩*
...	...	...

Concisely represent the PUC operator and the DC operator

Adapted from ArtInt

State	Action	Resulting State
<RLoc, RHC, SWC, MW, RHM>	puc	<RLoc, RHC, SWC, MW, RHM>
<RLoc, RHC, SWC, MW, RHM>	dc	<RLoc, RHC, SWC, MW, RHM>

Features to describe states

RLoc
- Rob's location (4-valued)

RHC
- Rob has coffee (binary)

SWC
- Sam wants coffee (binary)

MW
- Mail is waiting (binary)

RHM
- Rob has mail (binary)

Actions

mc
- move clockwise

mcc
- move counterclockwise

puc
- pickup coffee

dc
- deliver coffee

pum
- pickup mail

dm
- deliver mail

State	Action	Resulting State
⟨*lab*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩	*mc*	⟨*mr*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩
⟨*lab*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩	*mcc*	⟨*off*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩
⟨*off*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩	*dm*	⟨*off*, ¬*rhc*, *swc*, ¬*mw*, ¬*rhm*⟩
⟨*off*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩	*mcc*	⟨*cs*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩
⟨*off*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩	*mc*	⟨*lab*, ¬*rhc*, *swc*, ¬*mw*, *rhm*⟩
...	...	...

Concisely represent the PUC and DC operators

Adapted from ArtInt

State	Action	Resulting State
<cs, ~rhc, ?V5, ?V6, ?V7>	puc	<cs, rhc, ?V5, ?V6, ?V7>
<off, rhc, ?V8, ?V9, ?V10>	dc	<off, ~rhc, ~swc, ?V9, ?V10>

Features to describe states

RLoc
- Rob's location (4-valued)

RHC
- Rob has coffee (binary)

SWC
- Sam wants coffee (binary)

MW
- Mail is waiting (binary)

RHM
- Rob has mail (binary)

Exercise: specify a simple learning algorithm to generalize am operator description from the explicit state space representation

Why is generalization over mc instances different? Harder?

Actions

mc
- move clockwise

mcc
- move counterclockwise

puc
- pickup coffee

dc
- deliver coffee

pum
- pickup mail

dm
- deliver mail

State	Action	Resulting State
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨mr, ¬rhc,swc, ¬mw,rhm⟩*
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨off, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*dm*	*⟨off, ¬rhc,swc, ¬mw, ¬rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨cs, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨lab, ¬rhc,swc, ¬mw,rhm⟩*
...	...	...

STRIPS representation

Adapted from ArtInt

Features to describe states

RLoc
– Rob's location (4-valued)

RHC
– Rob has coffee (binary)

SWC
– Sam wants coffee (binary)

MW
– Mail is waiting (binary)

RHM
– Rob has mail (binary)

Actions

mc
– move clockwise

mcc
– move counterclockwise

puc
– pickup coffee

dc
– deliver coffee

pum
– pickup mail

dm
– deliver mail

State	Action	Resulting State
<cs, ~rhc, ?V5, ?V6, ?V7>	puc	<cs, rhc, ?V5, ?V6, ?V7>
<off, rhc, ?V8, ?V9, ?V10>	dc	<off, ~rhc, ~swc, ?V9, ?V10>

puc: Precondition {cs, ~rhc}; Effect {rhc}
dc: Precondition {off, rhc}; Effect {~rhc, ~swc}
mc-cs: Precondition {cs}; Effect {off}
mc-off: Precondition {off}; Effect {lab}
Mc-lab …; mc-mr …; mcc-cs …; mcc-mr …; mcc-lab …; mcc-off …; pum …; dm …;

State	Action	Resulting State
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨mr, ¬rhc,swc, ¬mw,rhm⟩*
⟨lab, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨off, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*dm*	*⟨off, ¬rhc,swc, ¬mw, ¬rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mcc*	*⟨cs, ¬rhc,swc, ¬mw,rhm⟩*
⟨off, ¬rhc,swc, ¬mw,rhm⟩	*mc*	*⟨lab, ¬rhc,swc, ¬mw,rhm⟩*
...	...	...

Initial State: {cs, ~rhc, swc, mw, ~rhm}

Goal State: {~swc}

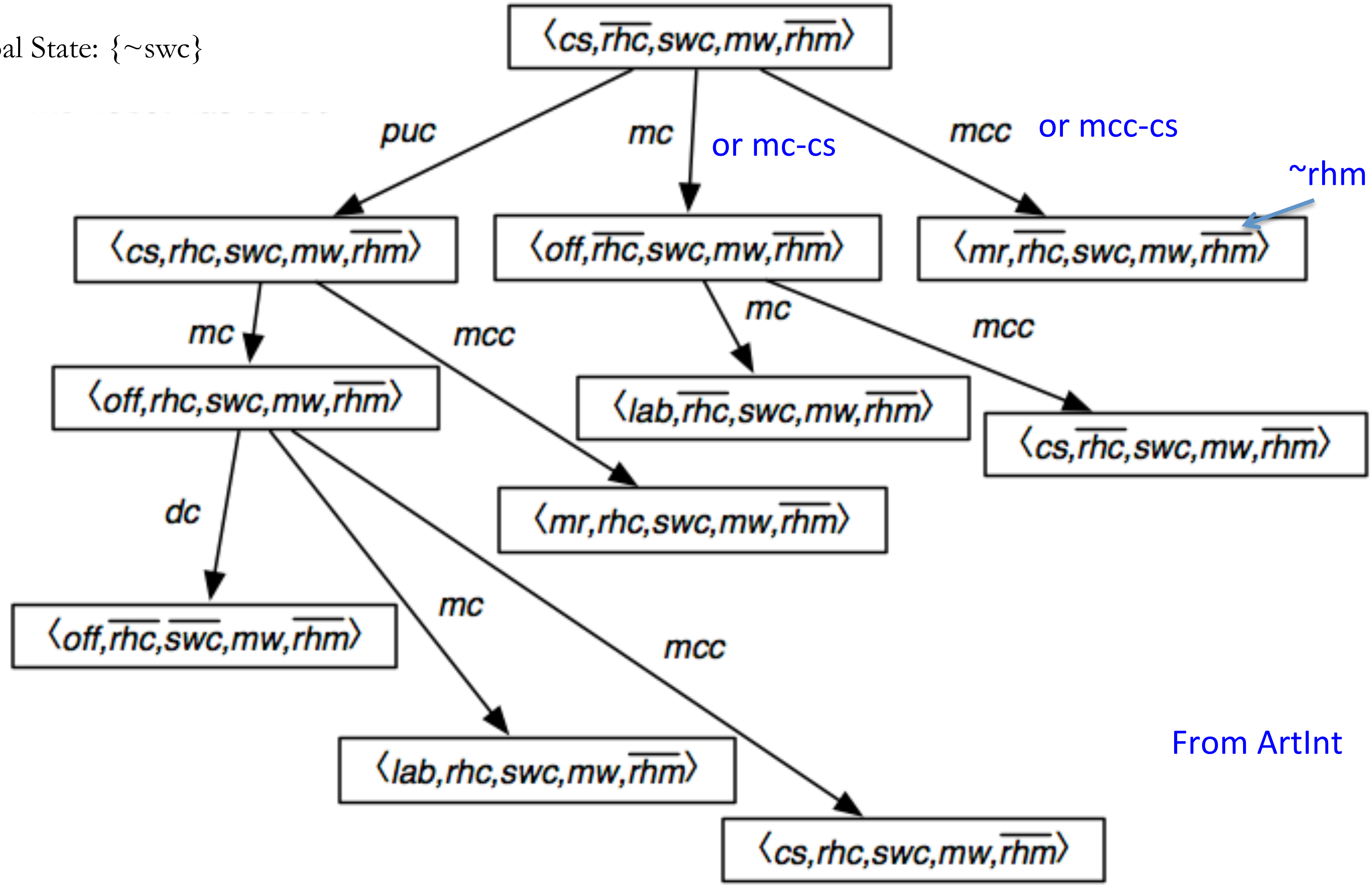
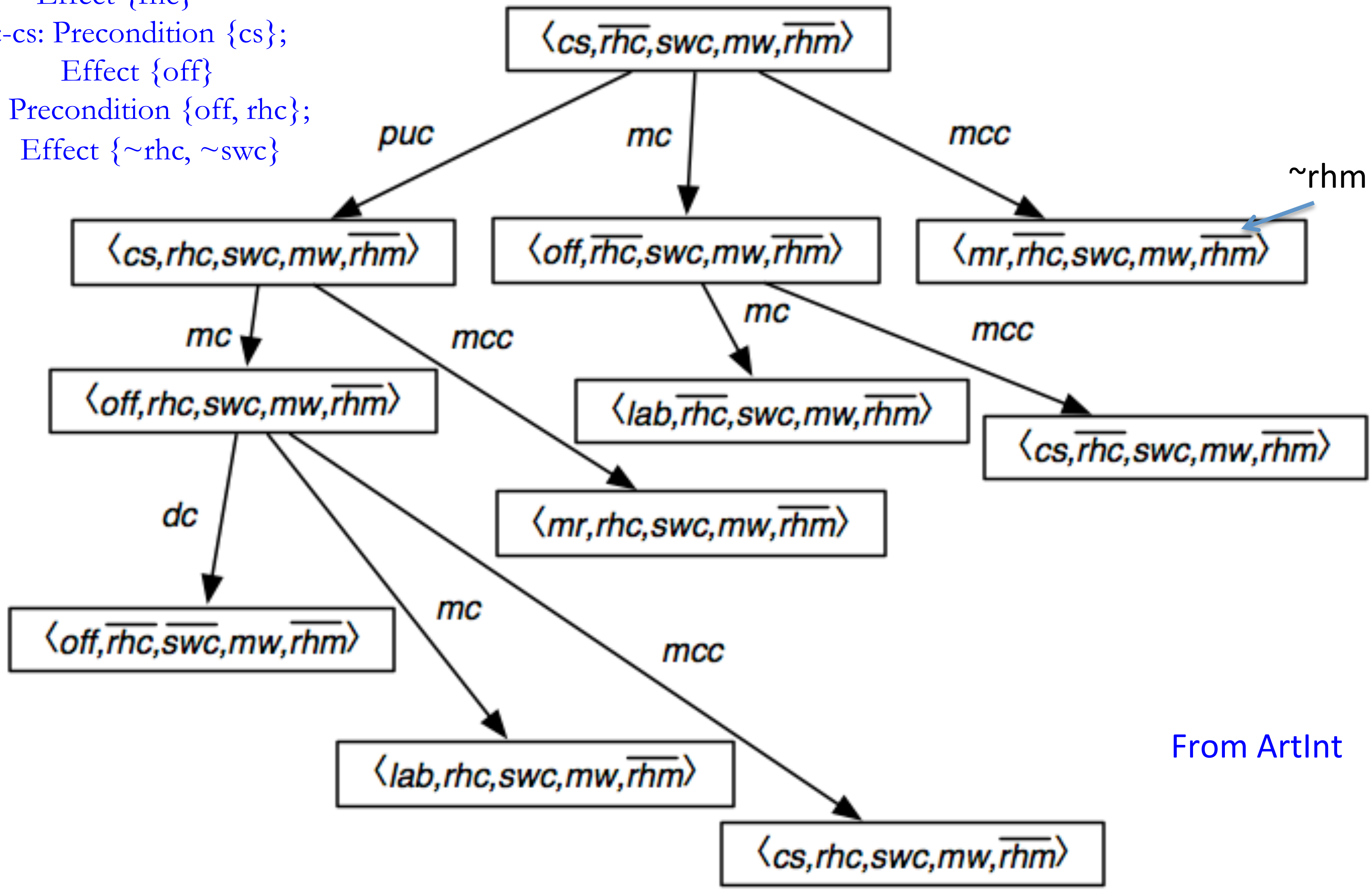


Figure 6.2 Part of the search space for a state-space planner

puc: Precondition {cs, ~rhc};
Effect {rhc}

mc-cs: Precondition {cs};
Effect {off}

dc: Precondition {off, rhc};
Effect {~rhc, ~swc}



From ArtInt

Figure 6.2 Part of the search space for a state-space planner

A depth-first forward search

puc: Precondition {cs, ~rhc};
Effect {rhc}
mc-cs: Precondition {cs};
Effect {off}
dc: Precondition {off, rhc};
Effect {~rhc, ~swc}

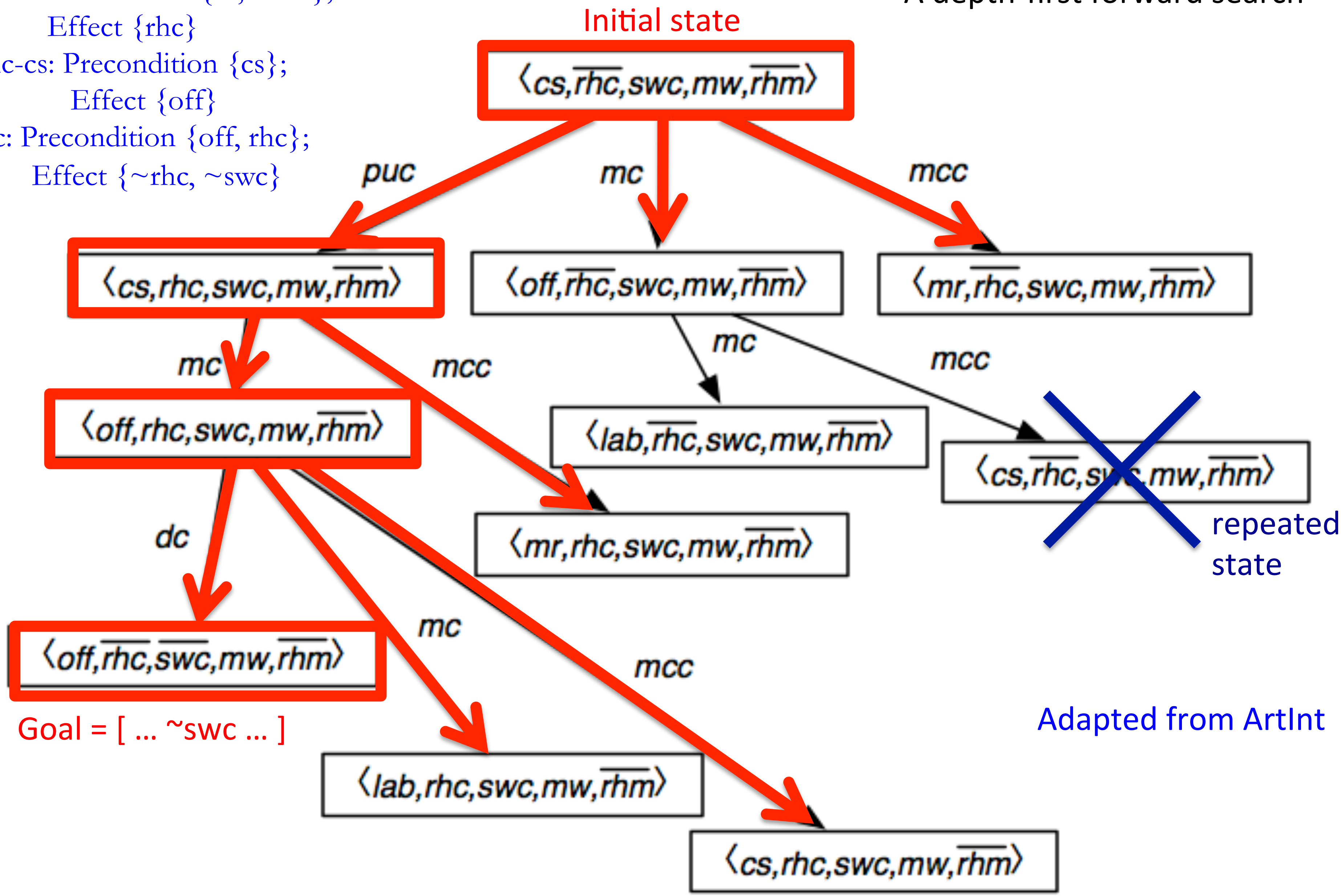


Figure 6.2 Part of the search space for a state-space planner

STRIPS Operators , which I will typically write pre(op) → eff(op)

puc: {RHC = ~rhc, RLOC = cs} → {RHC = rhc}

dc: {RHC = rhc, RLOC = off} → {RHC = ~rhc, SWC = ~swc}

mc_cs: {RLOC = cs} → {RLOC = off}

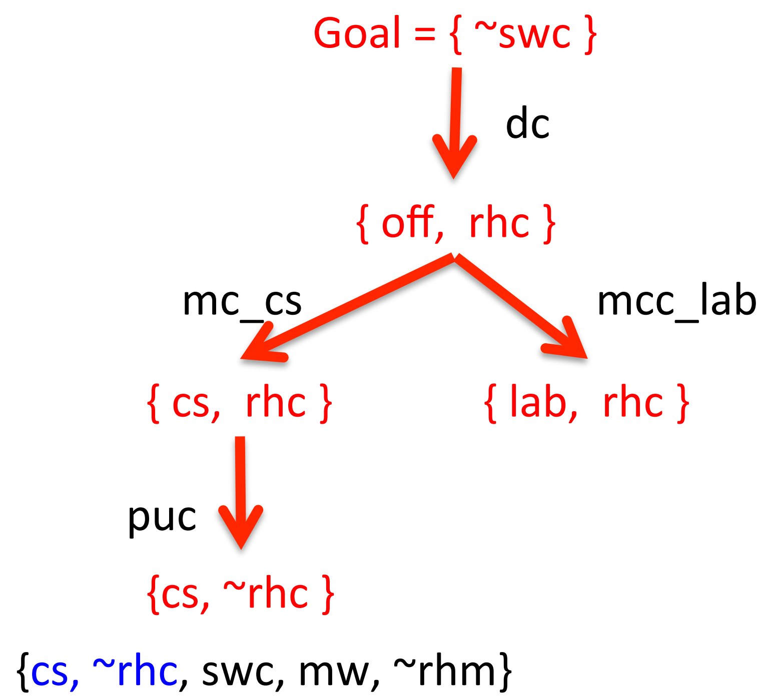
mcc_lab = {RLOC = lab} → {RLOC = off}

. . .

Initial State: {cs, ~rhc, swc, mw, ~rhm}

Goal State: {~swc}

Regression or backward planning



STRIPS Operators , which I will typically write pre(op) ➔ eff(op)

puc: {RHC = ~rhc, RLOC = cs} ➔ {RHC = rhc}

dc: {RHC = rhc, RLOC = off} ➔ {RHC = ~rhc, SWC = ~swc}

mc_cs: {RLOC = cs} ➔ {RLOC = off}

mcc_off = {RLOC = off} ➔ {RLOC = cs}

. . .

Exercise 6.6 from text

(c) puc;mc_cs

pre(puc; mc_cs) = ?   eff(puc; mc_cs) = ?

(d) puc; mc; dc

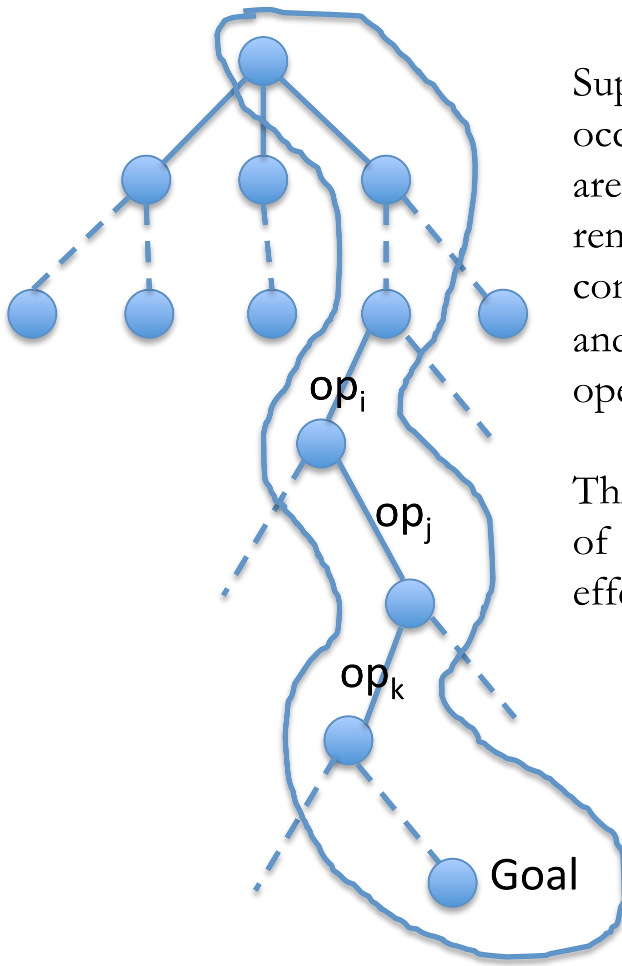
pre(puc;mc_cs; dc) = ?   eff(puc;mc_cs;dc) = ?

(e) mcc;puc;mc;dc

pre(mcc;puc;mc;dc) = ?   eff(mcc;puc;mc;dc) = ?

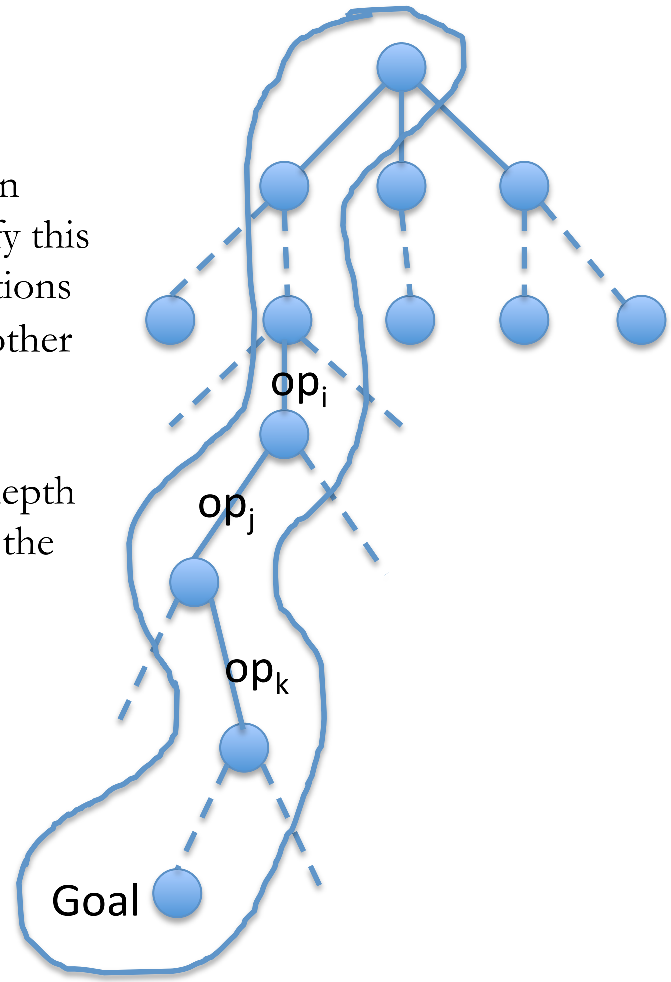
Why are composite (aka macro) operators useful?

Operators that frequently occur "back-to-back" may be useful to remember as a package



Suppose that op_i, op_j, and op_k occur frequently in plans that are found through search. Then remember op_i; op_j; op_k, identify this composite operators preconditions and effects, and treat like any other operator during search

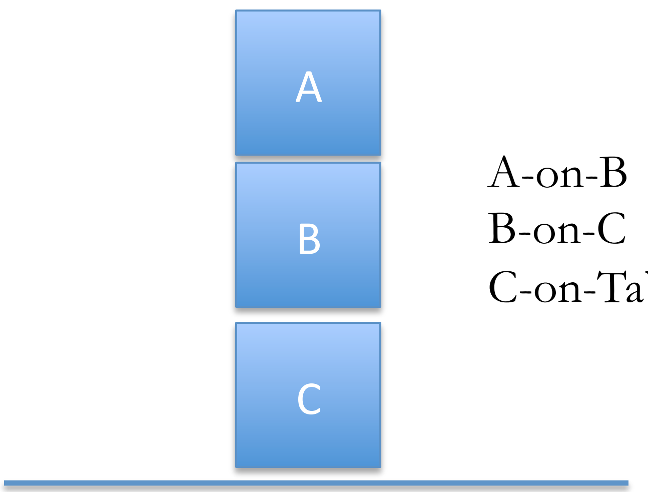
This can reduce the effective depth of search, but it also increases the effective breadth of search



Why are composite (aka macro) operators useful?

More interesting reason: macros can bridge places in the search where the heuristic is misleading

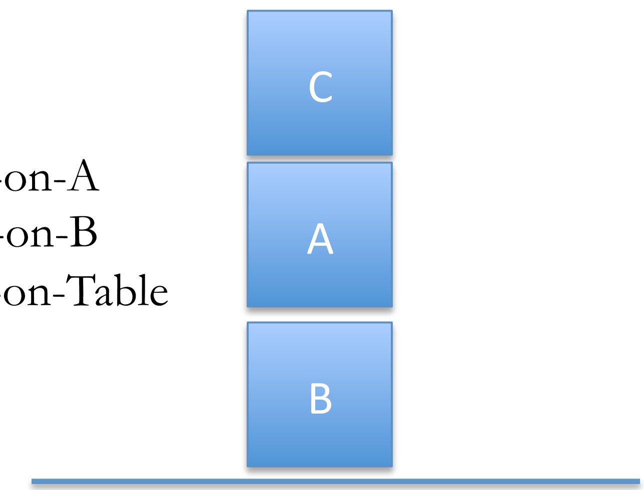
Consider this situation



A-on-B
B-on-C
C-on-Table

Initial State

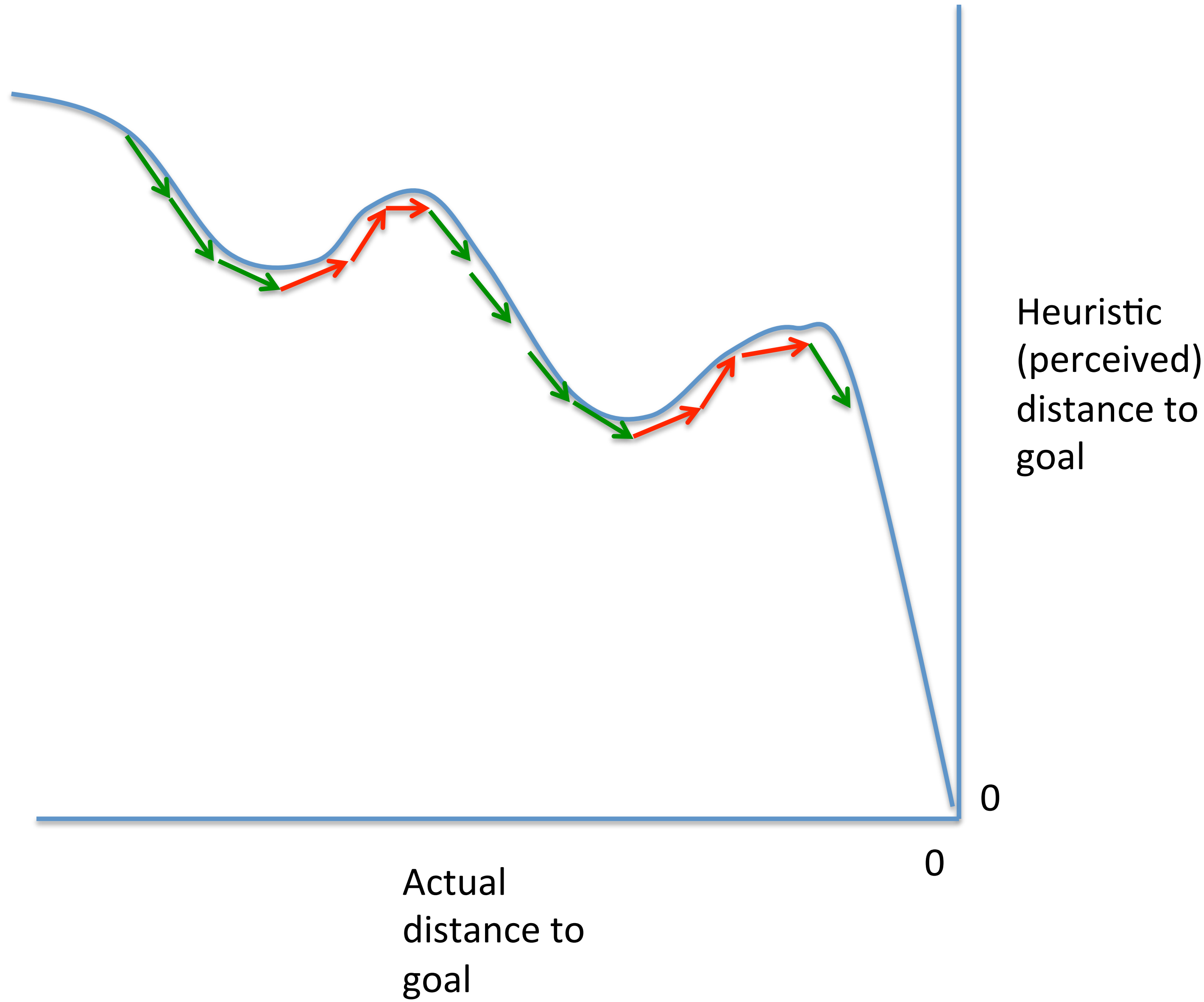
C-on-A
A-on-B
B-on-Table

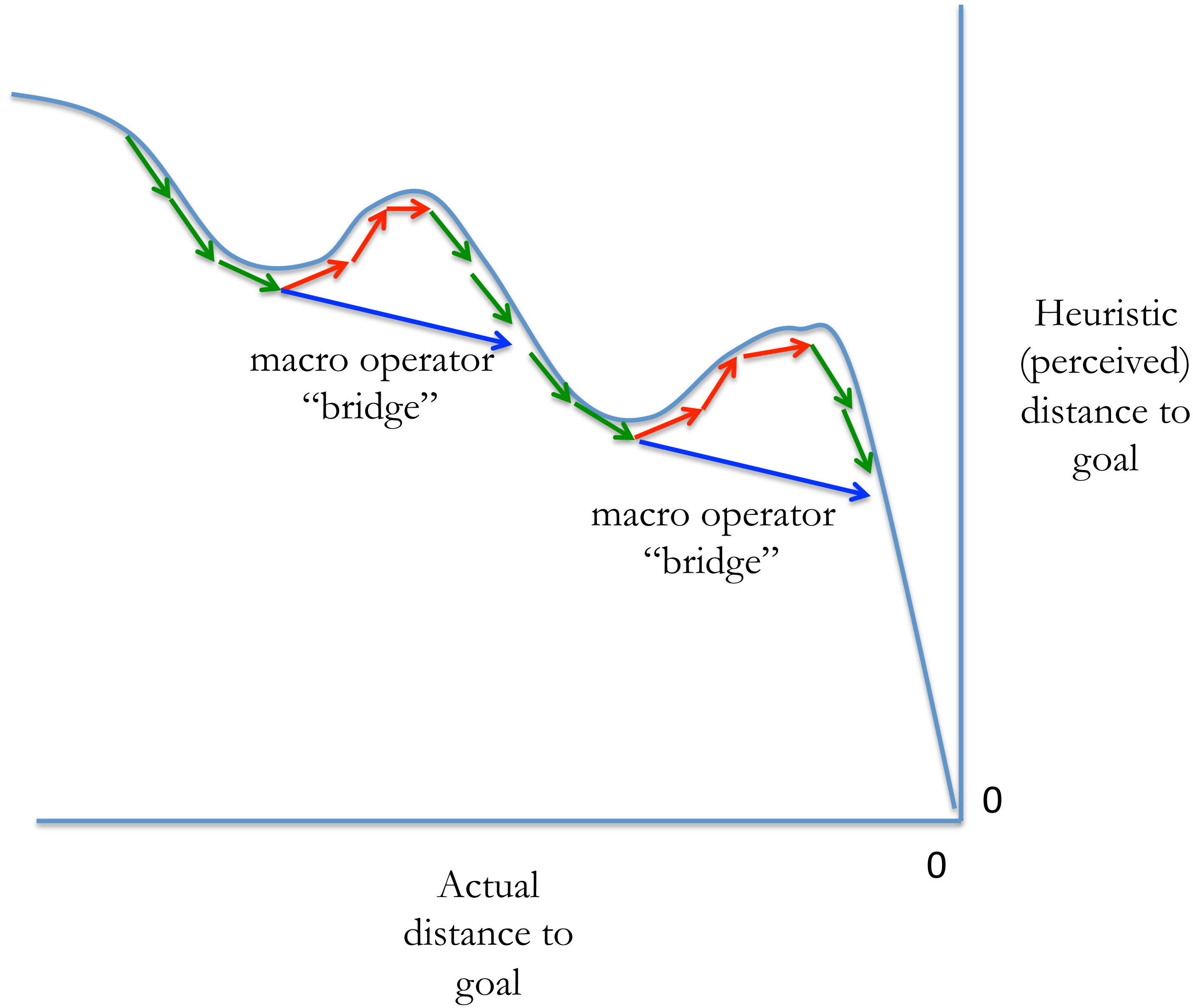


Goal State

Use forward search, with
heuristic that counts the number
of unachieved subgoals
so, h(Initial State) = 2

but it is necessary to use the unstack operator to remove A from B
to eventually achieve the final goal. This resulting intermediate state has an h value of 3


Heuristic (perceived) distance to goal
0
0
Actual distance to goal


macro operator
“bridge”
macro operator
“bridge”
Heuristic
(perceived)
distance to
goal
0
0
Actual
distance to
goal