Title: Planning & Acting
AIMA: Chapter 13 (Section 13.1)

Introduction to Artificial Intelligence
CSCE 476-876, Spring 2000
`www.cse.unl.edu/~cse476`

Berthe Y. Choueiry
`choueiry@cse.unl.edu, (402)472-5444`

Properties of abstract plans (Chapter 12)

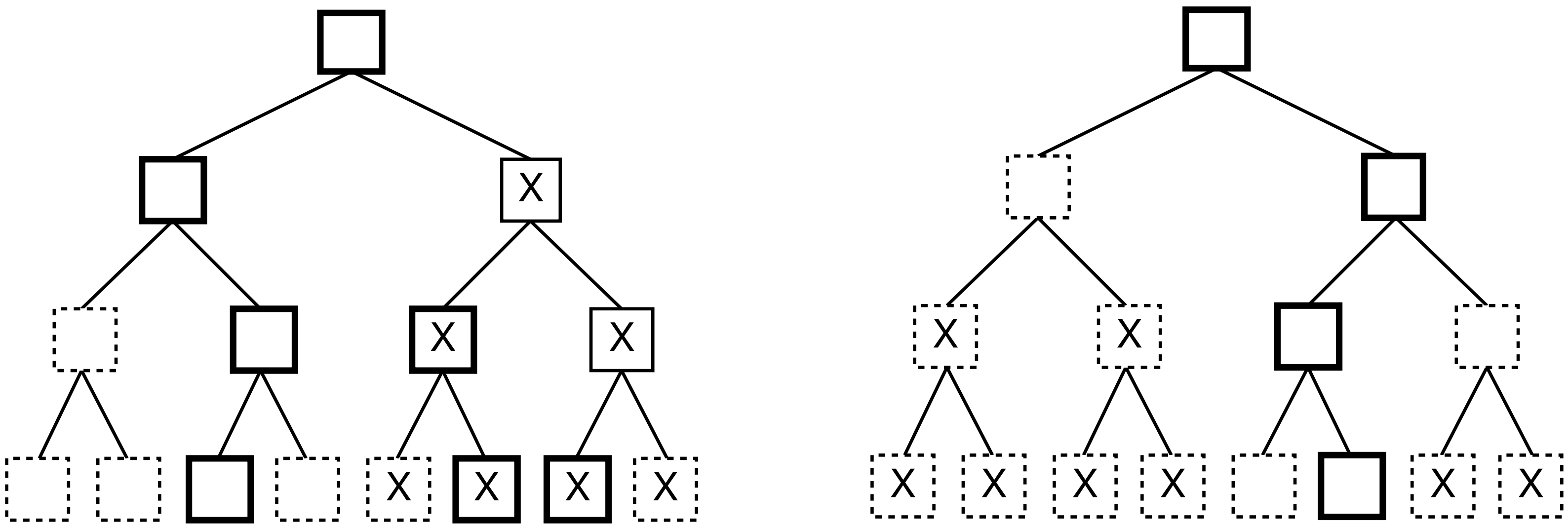
- Downward solution property: the existence of an abstract solution guarantees the existence of a primitive solution

  → the abstraction is a sufficient approximation
  → positive use of the abstraction

- Upward solution property (downward failure property): the existence of a primitive solution guarantees the existence of an abstract solution

  i.e., the non-existence of an abstract solution guarantees the non-existence of a primitive solution

  → the abstraction is a necessary approximation
  → negative use of the abstraction



(a) Downward Solution Property **(b) Upward Solution Property**

"Without these properties, or some reasonable substitute, a hierarchical planner does no better than a non-hierarchical planner in the worst-case (although it may do better in the average case)."

Planning and acting

Predictive planning (for flawless planning and execution):

- World: accessible, static, deterministic
- Actions' descriptions: correct and complete (*i.e.*, consequences specified exactly)

Real-world domains: incomplete and incorrect information

- Inaccessible world → incompleteness
- Dynamic world → incorrectness

Dealing with incompleteness and incorrectness

- **Conditional planning**: incomplete information

  Builds conditional plan that accounts for each possible situation

  Includes sensing actions in the plan to determine appropriate conditions

  Expensive, because it plans for many unlikely cases

- **Execution monitoring**: incorrect information

  Monitors plan execution

  Triggers <u>replanning</u> when conditions change (unforeseen events)

Watching for errors in execution planning

$\equiv$

adding error sensors in conditional planning

Conditional planning vs. Execution monitoring

→ Conditional planning foresees alternatives in case of errors

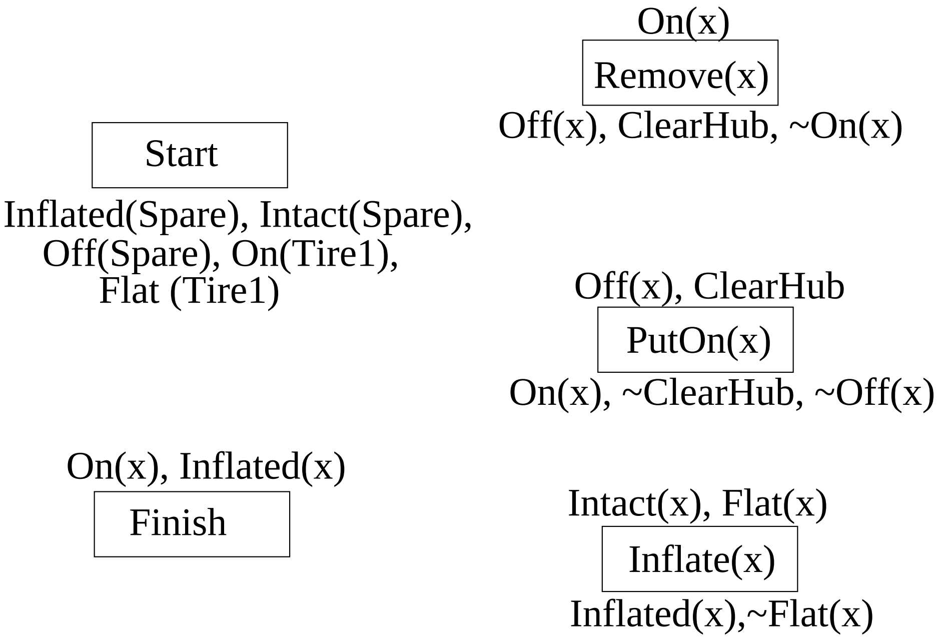
→ Execution planning defers dealing with errors until they arise

$$\text{In practice, combine} \begin{cases} \text{plan for some contingency} \\ \text{leave others until they occur} \end{cases}$$

Conditional plans

Problem: fixing a flat tire.

Initial conditions, goal, and operators:



Standard planner: $[Remove(Tire1), PutOn(Spare)]$

Things go wrong

Start

Inflated(Spare), Intact(Spare),
Off(Spare), On(Tire1),
Flat (Tire1)

On(x), Inflated(x)

Finish

Incomplete Information: $Intact(Tire1)$?

— Negation as failure: $\neg Intact(Tire1)$

— What if: $Punctured(Tire1)$? $NotPumped(Tire1)$?

$\rightarrow$ Conditional plan: If$(\langle Condition \rangle, \langle ThenPart \rangle, \langle ElsePart \rangle)$

If $Intact(Tire1), [Inflate(Tire1)], [Remove(Tire1), PutOn(Spare)]$

Conditional planning

$$[\ldots, \mathbf{If}(p, [then\, plan1], [else\, plan2]), \ldots]$$

Execution: check p against current KB, execute "then" or "else"

ThenPart and *ElsePart* may be plans themselves
$\longrightarrow$ any number of nested conditions..

Conditional planning: just like `POP` except
- if an open condition can be established by <u>observation</u> action
  - add the action to the plan
  - complete plan for each possible observation outcome
  - insert conditional link with these subplans

To ensure that a conditional plan is executable,

the agent must insert <u>actions</u> that cause

the relevant <u>conditions</u> to become known by the agent

$\boxed{\text{CheckTire(x)}}$

KnowsWhether(Intact(x))

Sensing actions

$$\boxed{\text{CheckTire(x)}}$$
KnowsWhether(Intact(x))

Sensing actions may have $\begin{cases} \text{‘Knowledge’ effects} \\ \text{Ordinary effects: check with water, } Wet(Tire) \end{cases}$

Sensing actions may have preconditions: Fetch a pump to check tire..

$\longrightarrow$ A conditional planner may need to create plans to obtain some needed information

Generating conditional plans

Essentially similar to POP..

Additional construct: Context

Context:

— union of conditions that must hold for a step to be executed

— branch in which the step lies

Example:

If $Intact(Tire1), [Inflate(Tire1)], [Remove(Tire1), PutOn(Spare)]$

Context of $Inflate(Tire1)$ is $(Intact(Tire1))$

→ All subsequent steps inherit the context

→ Contexts keep track of potential threats

→ Threats resolved by promotion, demotion and conditionning

Conditional planning Example (1)

Context of "Finish" step = (True)

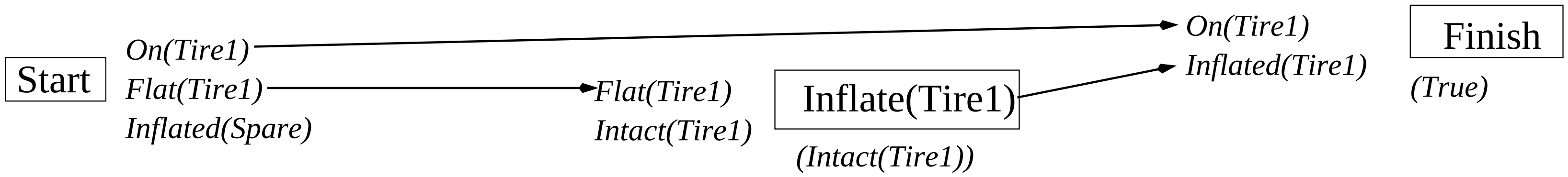
→ No assumptions made so far

Open conditions to be resolved: $On(x)$, $Inflated(x)$

Example (2)

$On(x)$: link to start step, unifier $\{x/Tire1\}$

$Inflated(x)$: add step $Inflate(Tire1)$



Step $Inflate(Tire1)$: two preconditions to resolve

$Flat(Tire1)$: link to start step

$Intact(Tire1)$?... no actions :—(

– POP fails.. attempts another plan..

– CPOP: $Check(Tire1)$

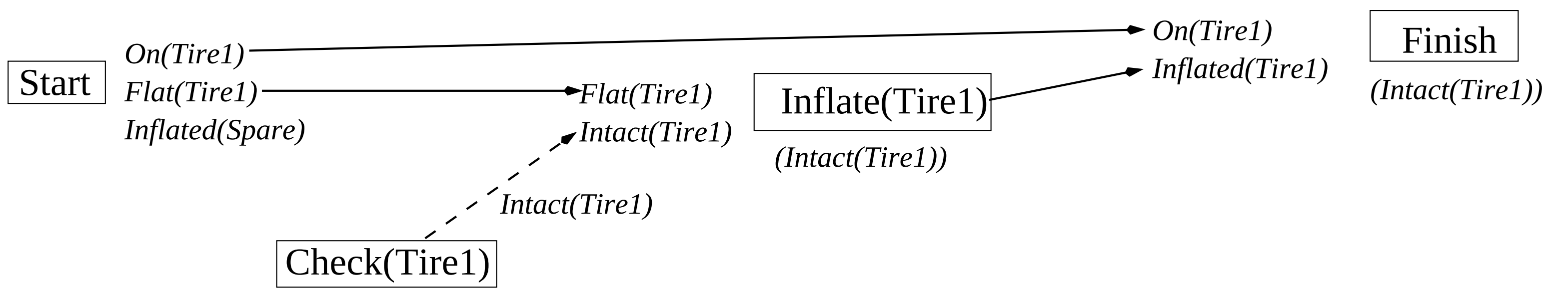
CheckTire(x)

KnowsWhether(Intact(x))

Example (3)

Condional Step $Check(Tire1)$ added with a **conditional link**
→ branching point in the final plan

$Inflate(Tire1)$ and $Finish$ acquire a new context

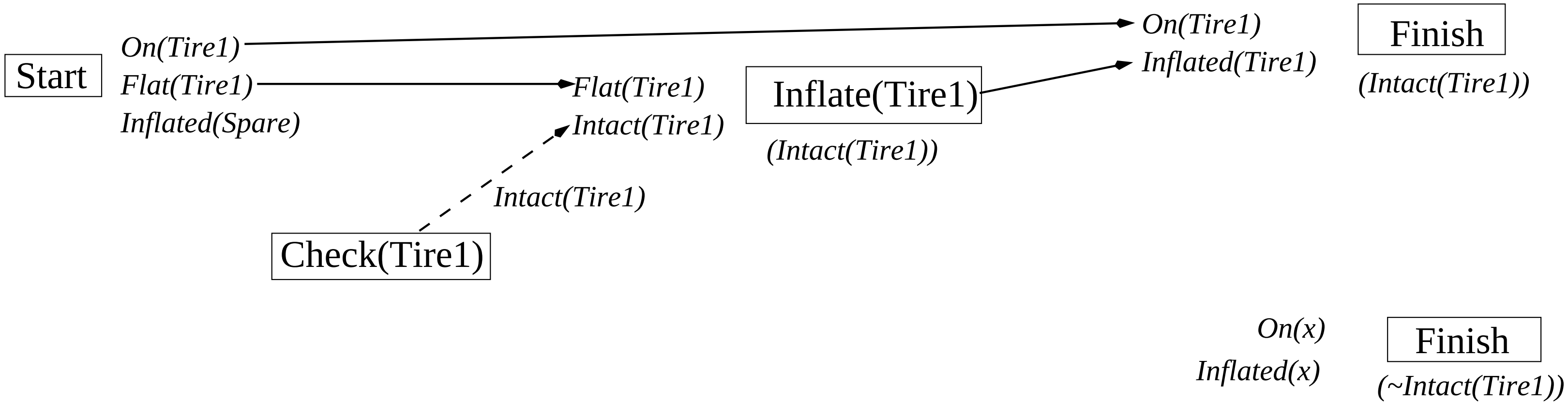


$Check(Tire1)$ has no precondition (HavePump), plan is complete

Done?.. NO!

Example (4)

Conditional planner adds a copy of $Finish$ step with negation of existing context
(for every possible outcome, there is a $Finish$ step)



Example (5)

Need to satisdy $Inflated(Tire1)$? Try $Inflate(Tire1)$:

Intact(x), Flat(x)

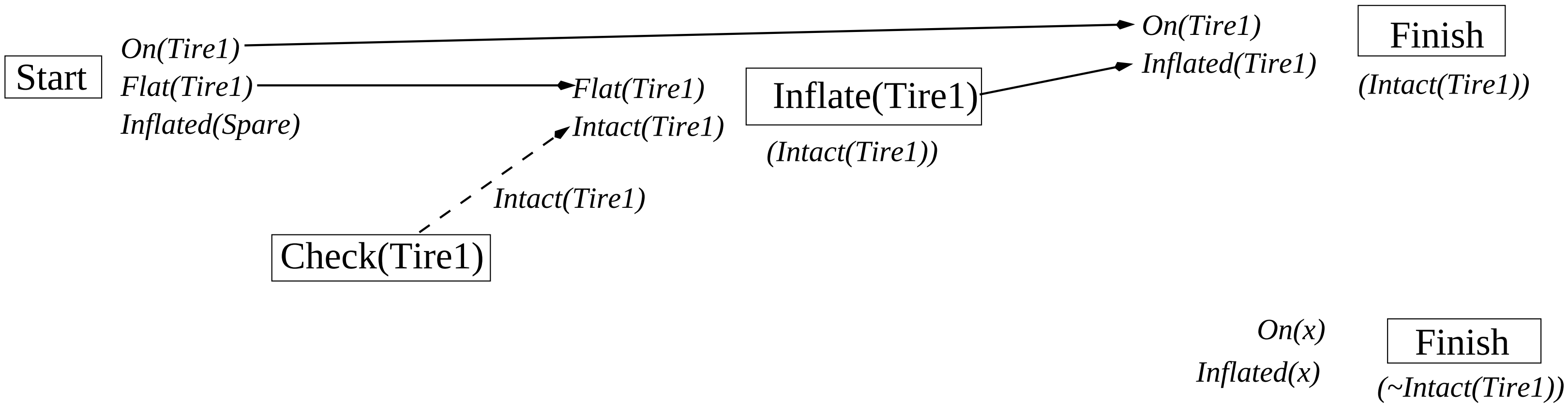
Inflate(x)

Inflated(x),~Flat(x)

Precondition $Intact(Tire1)$ is inconsitent with context

.. so, satisfy $Inflated(x)$ via:

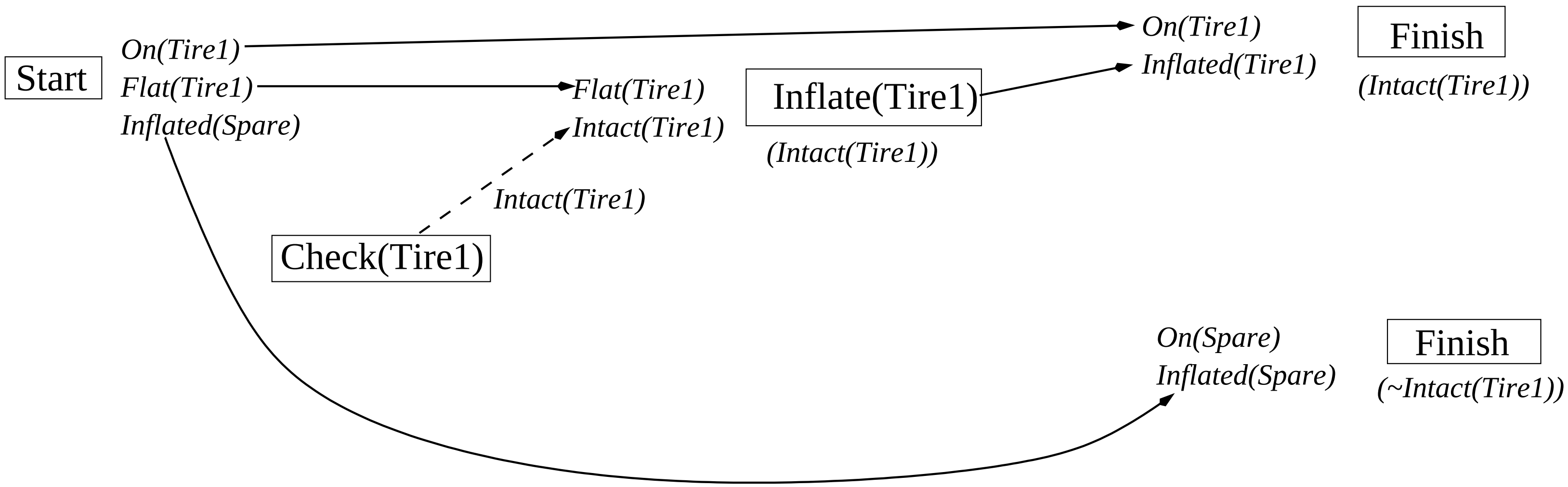
- $\surd$ link to $Start$ and unifier $\{x/Spare\}$, or
- $\times$ Add $Inflate(Spare)$ (dead-end)



Example (6)

Need to satisfy $On(Spare)$?

Add step $PutOn(Spare)$ then $Remove(Tire1)$, using causal links



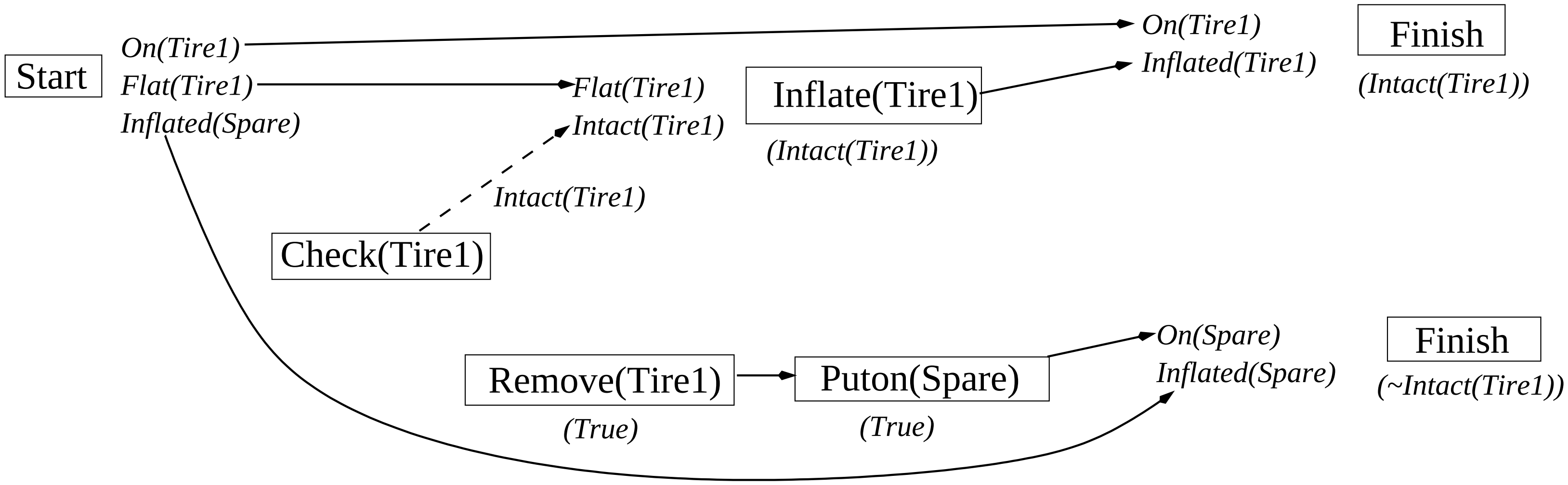
Example (7)

Initially: context is $(True)$

Check threats:

$Remove(Tire1)$ threatnes $On(Tire1)$ (with context $Intact(Tire1)$)

Threat resolution: promotion, demotion, or conditionning



Example (8)

Condition $Remove(Tire1)$ by making its context incompatible with that of the step it threatens ($i.e.$, first $Finish$)

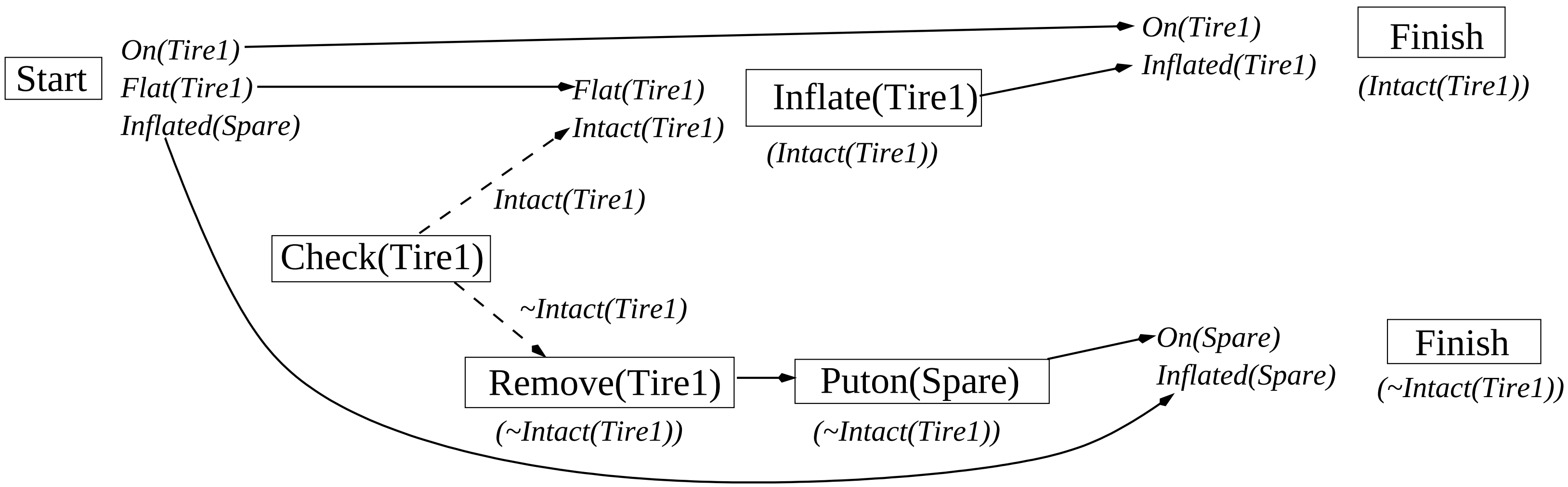
→ Find a conditional step that makes context of $Remove(Tire1)$ incompatible with that of (the first) $Finish$

→ $Check(Tire1)$ has a possible outcome $\neg Intact(Tire1)$

→ Add a conditional link between $Check(Tire1)$ and $Remove(Tire1)$

→ Propagate context to $PutOn(Tire1)$

→ Threat resolved :—)



CPOP algorithm

Builds on POP algorigthm and extends it by incorporating:

- contexts
- multiple finish steps
- conditioning process for threat resolution

function CPOP(*initial, goals, operators*) **returns** *plan*

plan ← MAKE-PLAN(*initial, goals*)
loop do

Termination:
if there are no unsatisfied preconditions
and the contexts of the finish steps are exhaustive
then return *plan*

Alternative context generation:
if the plans for existing finish steps are complete and have contexts $C_1 \ \ldots \ C_n$ **then**
add a new finish step with a context $\neg \, (C_1 \vee \ \ldots \ \vee \ C_n)$
this becomes the *current context*

Subgoal selection and addition:
find a plan step S_{need} with an open precondition c

Action selection:
choose a step S_{add} from *operators* or STEPS(*plan*) that adds c or
knowledge of c and has a context compatible with the current context
if there is no such step
then fail
add $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add $S_{add} \ < \ S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step **then**
add S_{add} to STEPS(*plan*)
add $Start \ < \ S_{add} \ < \ Finish$ to ORDERINGS(*plan*)

Threat resolution:
for each step S_{threat} that potentially threatens any causal link $S_i \xrightarrow{c} S_j$
with a compatible context **do**
choose one of
Promotion: Add $S_{threat} < S_i$ to ORDERINGS(*plan*)
Demotion: Add $S_j \ < \ S_{threat}$ to ORDERINGS(*plan*)
Conditioning:
find a conditional step S_{cond} possibly before both S_{threat} and S_j, where
1. the context of S_{cond} is compatible with the contexts of S_{threat} and S_j;
2. the step has outcomes consistent with S_{threat} and S_j, respectively
add conditioning links for the outcomes from S_{cond} to S_{threat} and S_j
augment and propagate the contexts of S_{threat} and S_j

if no choice is consistent
then fail
end
end

List of terms for glossary:

Due date: Monday, April 17, 2000

Words in margins in AIMA