

# Artificial Intelligence

## III Knowledge and Reasoning

### III.6 Planning

F. C. Langbein

School of Computer Science  
and Informatics  
Cardiff University



1.4

## Overview

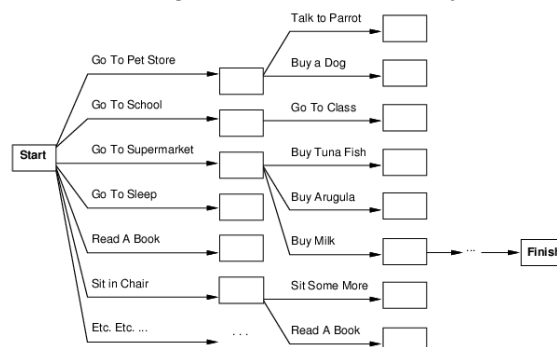
- Planning and searching
- Partial order planning
  - STRIPS operators
  - Planning graphs
- Planning and acting
  - Conditional planning
  - Monitoring and replanning

F. C. Langbein, Artificial Intelligence – III Knowledge and Reasoning; III.6 Planning

1

## Search vs. Planning

- Consider the task: *get milk, bananas, and a cord-less drill*
- Standard search algorithms fail miserably



- After-the-fact heuristic/goal test inadequate

F. C. Langbein, Artificial Intelligence – III Knowledge and Reasoning; III.6 Planning

2

## Search vs. Planning

- ▶ **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

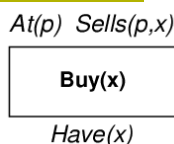
	Search	Planning
<b>States</b>	Data structures	Logical sentences
<b>Actions</b>	Code	Preconditions/outcomes
<b>Goal</b>	Code	Logical sentence (conjunction)
<b>Plan</b>	Sequence from $S_0$	Constraints on actions

## Planning in Situation Calculus

- ▶ **PlanResult**( $p, s$ ): situation resulting from executing  $p$  in  $s$ 
  - $\text{PlanResult}([], s) = s$
  - $\text{PlanResult}([a|p], s) = \text{PlanResult}(p, \text{PlanResult}(a, s))$
- ▶ **Initial state**:  $\text{At}(\text{Home}, S_0) \wedge \neg \text{Have}(\text{Milk}, S_0) \wedge \dots$
- ▶ **Actions**: successor state axioms
  - $\text{Have}(\text{Milk}, \text{PlanResult}(a, s)) \Leftrightarrow [(a = \text{Buy}(\text{Milk}) \wedge \text{At}(\text{Supermarket}, s)) \vee (\text{Have}(\text{Milk}, s) \wedge a \neq \dots)]$
- ▶ **Query**  
 $s = \text{PlanResult}(p, S_0) \wedge \text{At}(\text{Home}, s) \wedge \text{Have}(\text{Milk}, s) \wedge \dots$
- ▶ **Solution**  
 $p = [\text{Go}(\text{Supermarket}), \text{Buy}(\text{Milk}), \text{Buy}(\text{Bananas}), \dots]$

## STRIPS Operators

- ▶ Tidily arranged actions descriptions, restricted language:
  - ACTION**  $\text{Buy}(x)$ ,
  - PRECONDITION**:  $\text{At}(p) \wedge \text{Sells}(p, x)$ ,
  - EFFECT**:  $\text{Have}(x)$
  - Note, this abstracts away many important details!
- ▶ **Restricted** language  $\Rightarrow$  *efficient* algorithm
  - Precondition: **conjunction of positive literals**
  - Effect: **conjunction of literals**



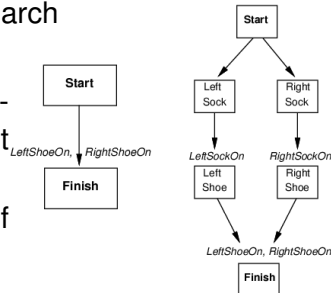
## State Space vs. Plan Space

- Standard search: node = concrete world state
- Planning search: node = **partial plan**
- **Open condition**: precondition of a step not yet fulfilled
- 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 with respect to another
- Gradually move from incomplete/vague plans to complete, correct plans

## Partially Ordered Plans

- Forward/backward state space search

- *Totally* ordered plan search
- Strictly *linear sequences* of actions directly connected to start or goal
- Cannot take advantage of *problem decomposition*



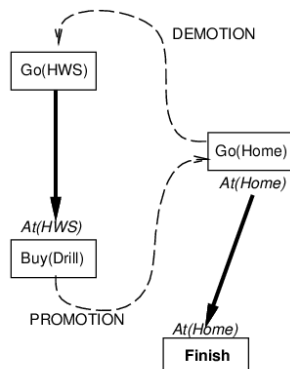
- **Partially Ordered Plans**

- 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

## Clobbering: Promotion & Demotion

- A **clobberer** is a potentially intervening step:

- Destroys condition achieved by a causal link
- E.g. Go(Home) clobbers At(HWS):



**Demotion**: put before Go(HWS)

**Promotion**: put after Buy(Drill)

## Partial-Order Planning

- A set of **actions/operators** that make up the steps of the plan
- A set of **ordering constraints**
- A set of **causal links**
- A set of **open preconditions**
- **Consistent plan**
  - No cycles in the ordering constraints
  - No conflicts with the causal links
- A **solution** is consistent plan with no open preconditions

## POP Algorithm Sketch

```
function plan ← POP (initial, goal, operators)
  plan ← MAKE-MINIMAL-PLAN(initial, goal)
  loop
    if SOLUTION (plan), return plan
    (Sneed, c) ← SELECT-SUBGOAL (plan)
    CHOOSE-OPERATOR (plan, operators, Sneed, c)
    RESOLVE-THREATS (plan)
```

```
function (Sneed, c) ← SELECT-SUBGOAL (plan)
  pick a plan step Sneed from STEPS (plan)
  with a precondition c that has not been achieved
  return (Sneed, c)
```

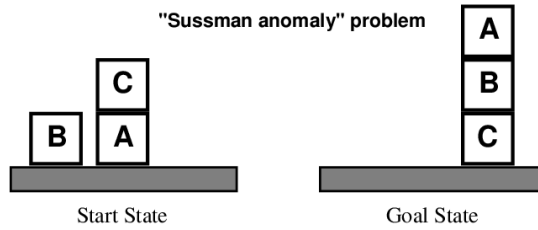
## POP Algorithm Sketch

```
procedure CHOOSE-OPERATOR(plan, operators, Sneed, c)
  choose step Sadd from operators or STEPS(plan)
  that has c as an effect
  if there no no such step, failure
  add causal link Sadd  $\xrightarrow{c}$  Sneed to LINKS(plan)
  add order constraint Sadd < Sneed to ORDERINGS(plan)
  if Sadd is newly added step from operators
    add Sadd to STEPS(plan)
    add Start < Sadd < Finish to ORDERINGS(plan)
```

```
procedure RESOLVE-THREATS(plan)
  for each Sthreat threatening a S1  $\xrightarrow{c}$  Sk in LINKS(plan)
  choose either
    DEMOTION: add Sthreat < S1 to ORDERINGS(plan)
    PROMOTION: add Sk < Sthreat to ORDERINGS(plan)
  if not CONSISTENT(plan), failure
```

# Example: Blocks World

"Sussman anomaly" problem



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

PutOn(x,y)

$\sim On(x,z) \ \sim Clear(y) \ Clear(z) \ On(x,y)$

$Clear(x) \ On(x,z)$

PutOnTable(x)

$\sim On(x,z) \ Clear(z) \ On(x, Table)$

+ several inequality constraints

# Example: Blocks World

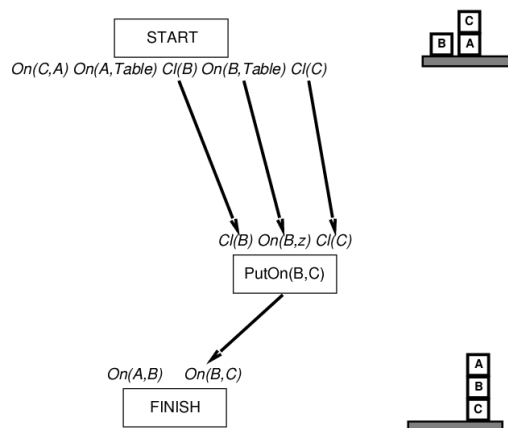


$On(A,B) \ On(B,C)$

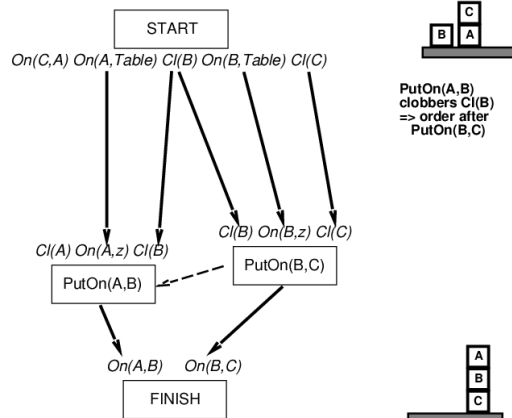
FINISH



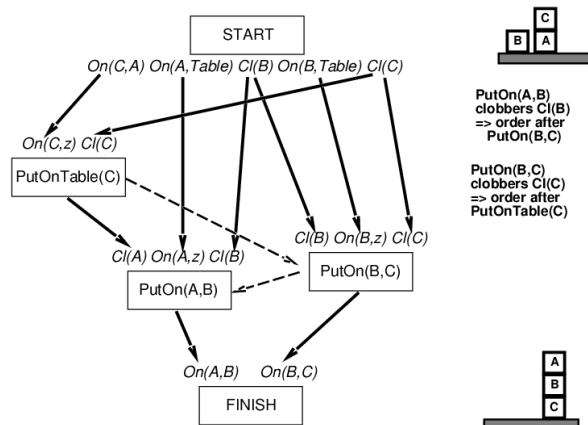
# Example: Blocks World



## Example: Blocks World



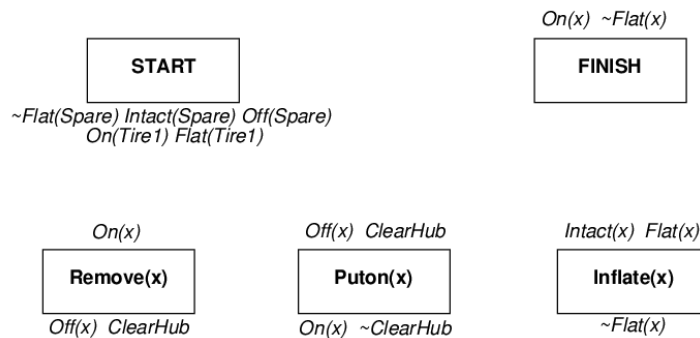
## Example: Blocks World



## POP Algorithm

- POP finds *shortest parallel plan*:
  - Sound, complete, systematic (no repetition)
  - Terminates with failure if no plan exists
- Issues:
  - No negated preconditions
  - No disjunction
  - No universal and existential quantifier
  - No conditional effects
- Currently not the most efficient method
  - Very sensitive to subgoal ordering

## The Real World



## Things Go Wrong

- **Incomplete information:**
  - Unknown preconditions, e.g.,  $Intact(Spare)?$
  - Disjunctive effects, e.g.,  $Inflate(x)$  causes  $Inflated(x) \vee SlowHiss(x) \vee Burst(x) \vee BrokenPump \vee \dots$
- **Incorrect information:**
  - Current state incorrect, e.g., spare NOT intact
  - Missing/incorrect postconditions in operators
- **Qualification problem:**
  - Can never finish listing all the required preconditions and
  - Possible conditional outcomes of actions

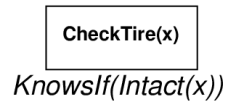
## Solutions

- **Conditional planning**
  - Plan to obtain information (**observation actions**)
  - Subplan for each contingency, e.g.,  $[Check(Tire1), If(Intact(Tire1), [Inflate(Tire1)], [CallAA])]$
  - *Expensive* because it plans for many unlikely cases
- **Monitoring/Replanning**
  - Assume normal states, outcomes
  - Check progress *during execution*, replan if necessary
  - Unanticipated outcomes may lead to failure (e.g. no AA card)
- In general, some monitoring is unavoidable

## Conditional Planning

[...] **If**( $p$ , [ $then\_plan$ ], [ $else\_plan$ ]), [...]

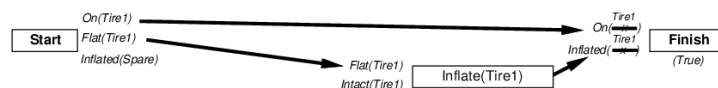
- Execution: check  $p$  against current  $KB$ , execute  $then\_plan$  or  $else\_plan$
- **Conditional planning**: just like POP except
  - If open condition can be established by *observation action*
    - Add the action to the plan
    - Complete plan for each possible observation outcome
    - Insert conditional step with these subplans



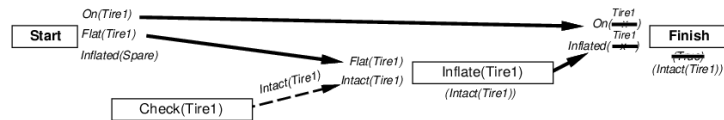
## Conditional Planning Example



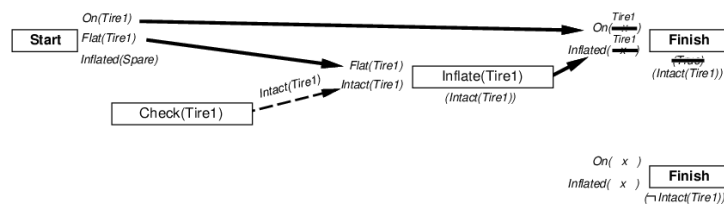
## Conditional Planning Example



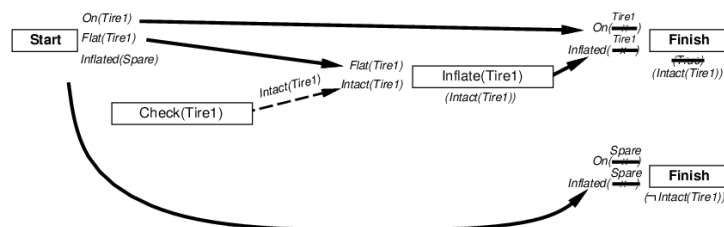
# Conditional Planning Example



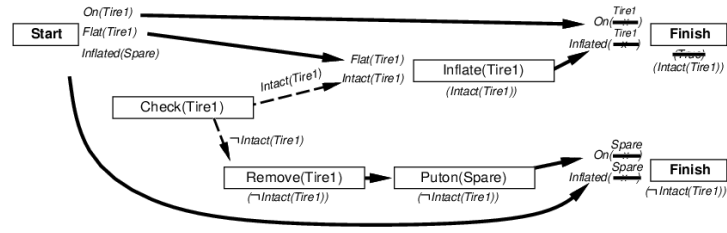
# Conditional Planning Example



# Conditional Planning Example



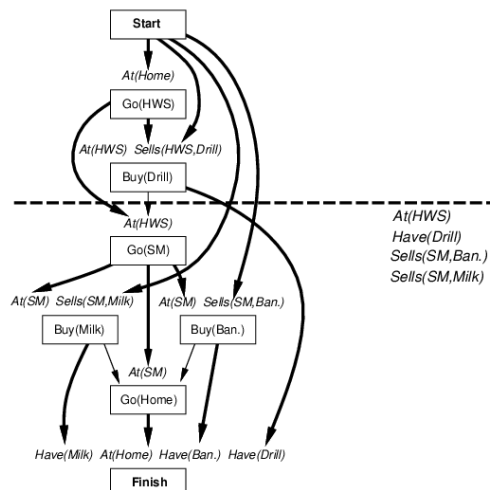
## Conditional Planning Example



## Monitoring

- **Execution monitoring**
  - “Failure” = preconditions of *remaining plan* not met
  - Preconditions = *causal links at current time*
- **Action monitoring**
  - “Failure” = preconditions of *next action* not met
    - or action itself fails, e.g., robot bump sensor fires
- In both cases, need to *replan*

## Preconditions for Remaining Plan



# Replanning

- Simplest: on failure, *replan from scratch*
- Better: plan to get back on track by *reconnecting to best continuation*
  - “Loop until done” behaviour without explicit loop

