Planning

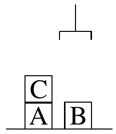
Planning involves:

- Given knowledge about a task domain
- Given a problem specified as:
  - an initial configuration of the state of the "world"
  - a set of goals to be achieved
- Find a **solution** to the problem, i.e., a *way* to transform the initial configuration into a new state of the world where the goal statement is true.
- Planning as Search: Action representation, multiple goals
- Different planning strategies – No universal dominance
- Different planning algorithms.

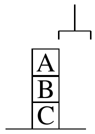
An Example in the Blocksworld

- *MOVE(x,y,z)* moves block x from the top of y to the top of z. y and z can be either the table or another block. *MOVE* is applicable only if x and z are clear, and x is on y. The table always has clear space. A block is clear if it does not have any other block on top of it.

Initial state:
(on B table)
(on A table)
(on C A)
(clear C)
(clear B)



Goal statement:
(and
(on A B)
(on B C))



Plan:
MOVE(C, A, table)
MOVE(B, table, C)
MOVE(A, table, B)

Many Issues To Resolve...

... A few are:

- What knowledge defines the task domain?
- How to represent the planning action model?
- What is the (sufficient) initial state of the world?
- What are the (prioritized) goals?
- How to acquire domain knowledge efficiently from expert human planners?
- What algorithm to generate the solution plan?
- How to generate plans in a computationally tractable way?
- How to create plans of *good* quality?
- How to scale up to real-world problems?

Domain Representation

- Operators – rules – with:
  - Precondition expression – must be satisfied before the operator is applied.
  - Set of effects – describe how the application of the operator changes the state.
- Precondition expression: propositional, typed first order predicate logic, negation, conjunction, disjunction, existential and universal quantification, and functions.
- Effects: add-list and delete-list.
- Conditional effects – dependent on condition on the state.

```
(ptype-of OBJECT :top-type)
(ptype-of BLOCK OBJECT)
(ptype-of TABLE OBJECT)

(OPERATOR MOVE
 (preconds
  ((<block> BLOCK)    ;;x
   (<from> OBJECT)    ;;y
   (<to> OBJECT))     ;;z
  (and (clear <block>)
       (clear <to>)
       (on <block> <from>)))
 (effects
   ((add (on <block> <to>))
    (del (on <block> <from>))
    (if (block-p <from>)
        (add (clear <from>)))
    (if (block-p <to>)
        (del (clear <to>))))))
```


Different Representations

```
(ptype-of OBJECT :top-type)

(OPERATOR PICK-UP
  (params <ob1>)
  (preconds
   ((<ob1> OBJECT))
   (and (clear <ob1>)
(on-table <ob1>)
(arm-empty)))
  (effects
   ()
   ((del (on-table <ob1>))
    (del (clear <ob1>))
    (del (arm-empty))
    (add (holding <ob1>)))))

(OPERATOR PUT-DOWN
  (params <ob>)
  (preconds
   ((<ob> OBJECT))
    (holding <ob>))
  (effects
   ()
   ((del (holding <ob>))
    (add (clear <ob>))
    (add (arm-empty))
    (add (on-table <ob>)))))
```



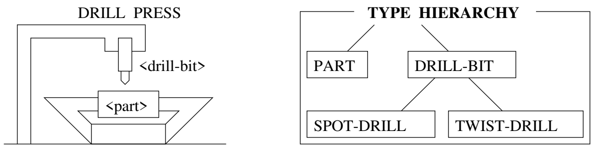
```
(OPERATOR STACK
  (params <ob> <underob>)
  (preconds
   ((<ob> Object)
    (<underob> OBJECT)
    (and (clear <underob>)
 (holding <ob>)))
  (effects
   ()
   ((del (holding <ob>))
    (del (clear <underob>))
    (add (clear <ob> ))
    (add (arm-empty))
    (add (on <ob> <underob>)))))

(OPERATOR UNSTACK
  (params <ob> <underob>)
  (preconds
   ((<ob> Object)
    (<underob> OBJECT)
   (and (on <ob> <underob>)
(clear <ob>)
(arm-empty)))
  (effects
   ()
   ((del (on <ob> <underob>))
    (del (clear <ob>))
    (del (arm-empty))
    (add (holding <ob>))
    (add (clear <underob>)))))
```


Planning Domains

Many AI planning domains with different degrees of *realism*:

- Process planning
- Image processing management
- Logistics transportation
- Crisis management
- Bank risk management
- Credit card fraud detection
- Robot navigation
- Blocks world
- Puzzles
- Matrix algebra
- Artificial domains
- ...

Domain representation



drill-spot (<part>, <drill-bit>)
<part>: type PART
<drill-bit>: type SPOT-DRILL
Pre: (holding-tool <drill-bit>)
(holding-part <part>)
Add: (has-spot <part>)

put-drill-bit (<drill-bit>)
<drill-bit>: type DRILL-BIT
Pre: tool-holder-empty
Add: (holding-tool <drill-bit>)
Del: tool-holder-empty

put-part(<part>)
<part>: type PART
Pre: part-holder-empty
Add: (holding-part <drill-bit>)
Del: part-holder-empty

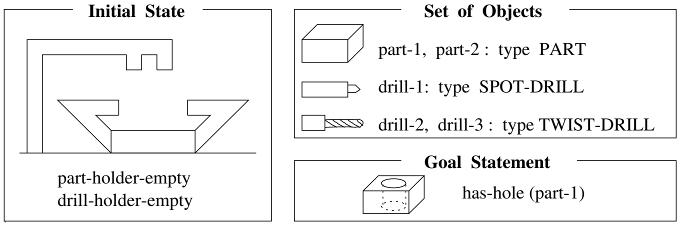
drill-hole(<part>, <drill-bit>)
<part>: type PART
<drill-bit>: type TWIST-DRILL
Pre: (has-spot <part>)
(holding-tool <drill-bit>)
(holding-part <part>)
Add: (has-hole <part>)

remove-drill-bit(<drill-bit>)
<drill-bit>: type DRILL-BIT
Pre: (holding-tool <drill-bit>)
Add: tool-holder-empty
Del: (holding-tool <drill-bit>)

remove-part(<part>)
<part>: type PART
Pre: (holding-part <drill-bit>)
Add: part-holder-empty
Del: (holding-part <drill-bit>)

A simplified version of the Process Planning domain.

Problem



A problem in the simplified Process Planning domain.

Plan

```
put-part(part-1)
put-drill-bit(drill-1)
drill-spot(part-1, drill-1)
remove-drill-bit(drill-1)
put-drill-bit(drill-2)
drill-hole(part-1, drill-2)
```

First operators in the example partial orden plan.

Search Strategies

Using a set of operators:

- Forward chaining – progression
  - From the initial state,
  - Apply operators with preconditions true in the state,
  - Get new states.
- Backward-chaining – regression
  - From the goal state,
  - Find operators that can add goal,
  - Set its preconditions as new goals.
- Partial order – network of constraints among plan steps – no direct reasoning about an explicit state.
- Total order – plan steps are ordered during search – use of a uniquely specified state.

Generating a Solution Plan

Several planning **algorithms**:

- Linear planning – Planning with a goal **stack**.
- Nonlinear planning – Interleaving of goals
  - State-space search
  - Plan-space search
- Hierarchical planning
  - Emphasis on action decomposition/refinement
  - Very little search

A complex process:

- **Alternative operators to achieve a goal.**
- **Multiple goals that interact.**
- **Efficiency, quality, and accuracy – hard.**

Means-ends Analysis

[**Newell and Simon 60s**] [**Ernst and Newell 69**]

GPS Algorithm (*initial-state, goals*)

- If *goals* $\subseteq$ *initial-state*, then return *True*
- Choose a difference d between *initial-state* and *goals*
- Choose an operator o to reduce the difference d
- If no more operators, then return *False*
- *State*=**GPS**(*initial-state*, preconditions(o))
- If *State*, then return **GPS**(apply(o,*State*), *goals*)

Example: Irreversible Actions

[Veloso 89]

```
(OPERATOR LOAD-ROCKET
 (preconds
  ((<roc> ROCKET)
   (<obj> OBJECT)
   (<loc> LOCATION))
  (and (at <obj> <loc>)
       (at <roc> <loc>)))
 (effects ()
  (add (inside <obj> <roc>))
  (del (at <obj> <loc>))))
```

```
(OPERATOR UNLOAD-ROCKET
 (preconds
  ((<roc> ROCKET)
   (<obj> OBJECT)
   (<loc> LOCATION)))
  (and (inside <obj> <roc>)
       (at <roc> <loc>)))
 (effects ()
  (add (at <obj> <loc>))
  (del (inside <obj> <roc>))))
```

```
(OPERATOR MOVE-ROCKET
 (preconds
  ((<roc> ROCKET)
   (<from-l> LOCATION)
   (<to-l> LOCATION))
  (and (at <roc> <from-l>)
       (has-fuel <roc>)))
 (effects ()
  (add (at <roc> <to-l>))
  (del (at <roc> <from-l>))
  (del (has-fuel <roc>))))
```


Incompleteness of Linear Planning

Initial state:
(at obj1 locA)
(at obj2 locA)
(at ROCKET locA)
(has-fuel ROCKET)

Goal statement:
(and
(at obj1 locB)
(at obj2 locB))

Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA) (MOVE-ROCKET) (UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	*failure*

Goal	*Plan*
(at obj2 locB)	(LOAD-ROCKET obj2 locA) (MOVE-ROCKET) (UNLOAD-ROCKET obj2 locB)
(at obj1 locB)	*failure*

State-Space Nonlinear Planning

Extend linear planning:

- From **stack** to **set** of goals.
- Include in the search space all possible interleaving of goals.

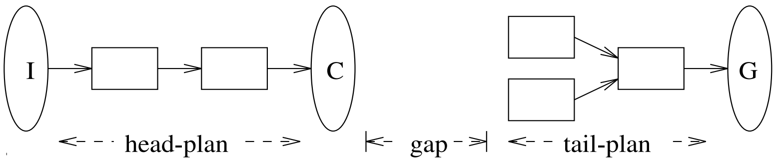
State-space nonlinear planning is **complete**.

Goal	*Plan*
(at obj1 locB)	(LOAD-ROCKET obj1 locA)
(at obj2 locB)	(LOAD-ROCKET obj2 locA)
	(MOVE-ROCKET)
(at obj1 locB)	(UNLOAD-ROCKET obj1 locB)
(at obj2 locB)	(UNLOAD-ROCKET obj2 locB)

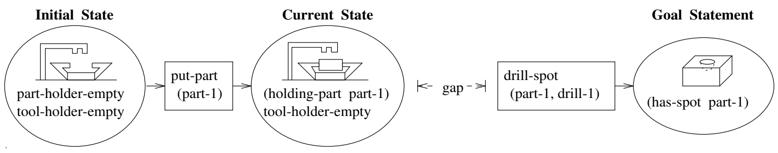
PRODIGY4.0 Planning Algorithm

1. Terminate if the goal statement is satisfied <u>in the current state</u>.
2. Compute the **SET** of *pending goals* $\mathcal{G}$, and the **set** of *applicable operators* $\mathcal{A}$.
   - A goal is pending if it is a precondition, not satisfied in the current state, of an operator already in the plan.
   - An operator is applicable when all its preconditions are satisfied in the state.
3. Choose a goal G in $\mathcal{G}$ or choose an operator A in $\mathcal{A}$.
4. If G has been chosen, then
   - *Expand goal* G, i.e., get the set $\mathcal{O}$ of *relevant instantiated operators* that could achieve the goal G,
   - Choose an operator O from $\mathcal{O}$,
   - Go to step 1.
5. If an operator A has been selected as directly applicable, then
   - *Apply* A,
   - Go to step 1.

Plan representation

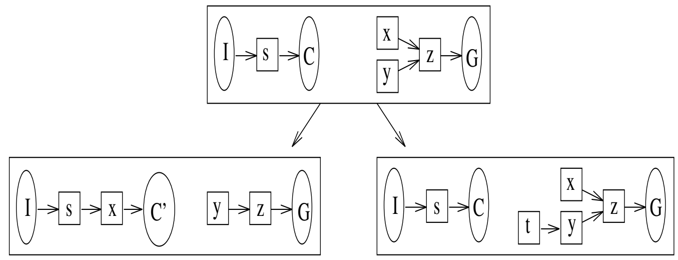


Representation of an incomplete plan.



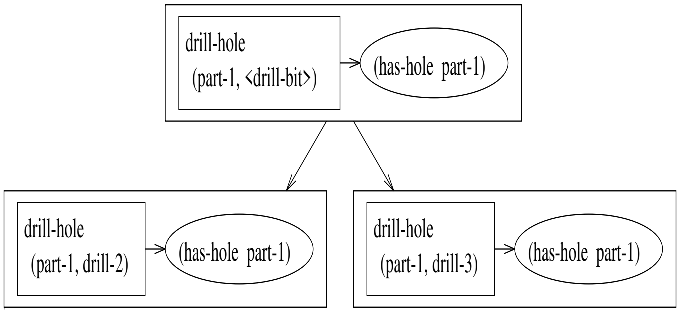
Example of an incomplete plan.

Plan Modifications



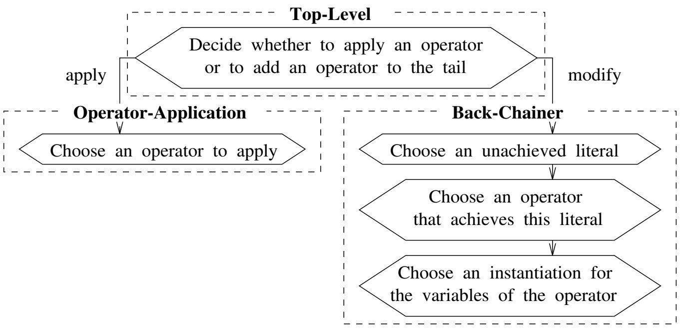
Applying an operator (moving it to the head) Adding an operator to the tail-plan

Modifying the current plan.



Instantiating a newly added operator.

Planning Choices



Branching decisions.

Planning and Learning

- Planning as search, i.e., a decision-making process – learn search heuristics
- Planning representation – learn efficient, complete, correct domain specifications

Control Rules - Heuristic to Guide Search

Select Rule

If (has-spot <part>) is the current goal
and drill-spot (<part>, <drill>) is the current operator
and (holding-drill-bit <drill-1>) holds in the current state
and <drill-1> is of the type SPOT-DRILL
Then select instantianting <drill> with <drill-1>

Prefer Rule

If (has-hole <part-1>) is a candidate goal
and (has-hole <part-2>) is a candidate goal
and (holding-part <part-1>) holds in the current state
Then prefer the goal (has-hole <part-1>) to (has-hole <part-2>)

Several Other Planning Algorithms

- TWEAK [Chapman 87], SNLP [McAllester & Rosenblitt 91] , UCPOP [Penberthy and Weld 92]
  - Emphasis on plan-space *search*
- NONLIN [Tate 76], O-PLAN [Tate], SIPE [Wilkins 88]
  - Emphasis on plan decomposition
- UNPOP, Planning and acting [McDermott 78]
- Reactive planning [Georgeff & Lansky 87], [Firby 87], [Hendler & Sanborn 87]
- Action and time [Allen 84] [Dean & McDermott 87]
- Walksat [Selman et al. 92, Kautz & Selman 92, 96]
- Flecs [Veloso & Stone 95]
- Graphplan [Blum & Furst 95]