Chapter MK:VI

VI.   Planning and Configuration

Planning Algorithms
Properties

❏ Soundness

A planning algorithm is sound if the plans found are solution plans for the given planning problem, i.e., the plan is applicable in the initial state and can lead to a state that satisfies the goal.

❏ Completeness

A planning algorithm is complete when it finds a plan that solves the planning problem, if such a plan exists.

❏ Optimality

A planning algorithm is optimal if the found plans are optimal (e.g., minimal with respect to length or cost) among all plans that solve the planning problem.

(Optimality is usually neglected.)

Planning Algorithms
Approaches and Restrictions

❑ **Linear Planning vs. Non-linear Planning**

Goals are processed one after the other in a fixed order. In non-linear planning, goals can be interleaved (interleaving, goal set instead of goal stack).

Linear planning is possible if goals can be achieved independently of each other, i.e. no operation to reach a goal blocks other goals.
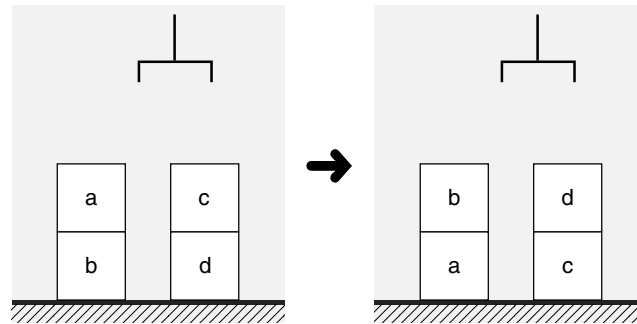
❑ **Progression vs. Regression**

Search for a plan can be carried out forward from the initial state or backward from a goal.
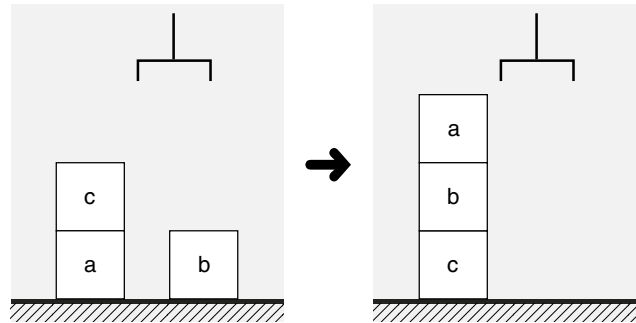
Planning Algorithms

Approaches and Restrictions (continued)

❑ Totally Ordered Plans vs. Partially Ordered Plans

Plans can be constructed as sequences of actions. However, plans can also be created as linearizations of a set of actions with order constraints, whereby the number of actions and constraints are gradually increased.

Not every total order of operations from partial plans to reach individual goals lead to the overall goal, in particular, subplans must be interleaved if necessary.

❑ State Space Search vs. Plan Space Search

In a state space, the search space graph is formed by states with actions as edges, in the plan space, the search space graph consists of plans with plan transformations as edges.

Planning Algorithms
Planning as State Space Search

❑ The search space graph is formed by

  – state resp. sets of states (given by state descriptions) as nodes and

  – applicable actions (instances of operators) as edges defining the successor states.

❑ A plan is defined by a directed path from the initial state to a state that satisfies the required goal.

❑ Examples of planning algorithms

  – Forward Search (starting from the initial state in direction to goal states) (Forward Planning, Forward Search, Progression Planning)

  – Backward Search (starting from the goal in direction to a description satisfied by the initial state) (Backward Planning, Backward Search, Regression Planning)

  – Recursive STRIPS-Planning

Planning Algorithms
Planning as Plan Space Search

❑ The search space graph is formed by

– partial plans (set of partially instantiated operators and order constraints for operators) as nodes and

– possible extensions of partial plans as edges.

❑ A linear plan can be generated from the information in a target node.

❑ Examples of planning algorithms

– Partial-Order Planning

Planning Algorithms
HTN Planning

❑ HTN Planning aims to process a set of tasks, taking into account constraints for the order and properties of the states.

❑ The search space graph is formed by

- hierarchical task networks as nodes and
- network transformations by task decompositions as edges.

❑ Examples of planning algorithms

- Total-Order Forward Decomposition

Remarks:

❑ Further reading:

    – Jörg Hoffmann. *Everything You Always Wanted to Know About Planning (But Were Afraid to Ask)*, in: Proceedings of the 34th Annual German Conference on Artificial Intelligence (KI'11), 2011 (PDF)

Chapter MK:VI

VI. Planning and Configuration

State-Space Planning
Forward Search in State Spaces (Progression Planning)

To create a plan for a goal, we look at the initial state and try to reach a state that satisfies the goal by using actions.

Nondeterministic algorithm:

1. Start with an empty plan p from the initial state s_{init}.

2. While the current state does not satisfy the goal conditions g do:

    (a) If no action can be applied, then return *Failure*.

    (b) Choose an applicable action (i.e. an operator with a ground instantiation that forms an applicable action).

    (c) Add the action to the end of the plan p.

    (d) Determine the successor state of the action and use it as new current state.

3. Return the plan p.

State-Space Planning

Forward Search in State Spaces (Progression Planning)   (continued)

- ❏ Forward Search is *sound*.

  If a plan is returned , this plan solves the planning problem at hand.

- ❏ Forward Search is *complete*.

  If a plan exists that is a solution to the planning problem, then such a plan can be found by the non-deterministic algorithm.

- ❏ Deterministic implementations of Forward Search:

  - Breadth-First Search
    (sound and complete; terminates if cycles are pruned; space requirement exponential in size of problem)
  - Depth-First Search
    (sound; complete when used with iterative deepening or cycles are pruned; space requirement linear in size of problem)
  - Greedy Search
    (sound; usually not complete; space requirement linear in size of problem)
  - Best-First Search (BF*, A*)
    (sound; complete under certain restrictions; space requirement exponential in size of problem)

Remarks:

❑ From a search perspective, there is no need to explicitly represent a plan, since plans correspond to the backpointer paths. The applicability of actions in a plan is guaranteed by construction.

❑ Examples of forward search planning systems:

– Blair Bonet, Hugh Geffner. *HSP: Heuristic Search Planner*, in:AIPS-98 Planning Competition (1998)
– Burkhard Nebel. *The FF Planning System: Fast Plan Generation Through Heuristic Search*, in: Journal of Artificial Intelligence Research 14 (2001), pp. 253-302
– Malte Helmert. *The Fast Downward Planning System*, in: Journal of Artificial Intelligence Research 26 (2006), pp. 191-246

State-Space Planning

Example: Forward Search in Blocks World

State space
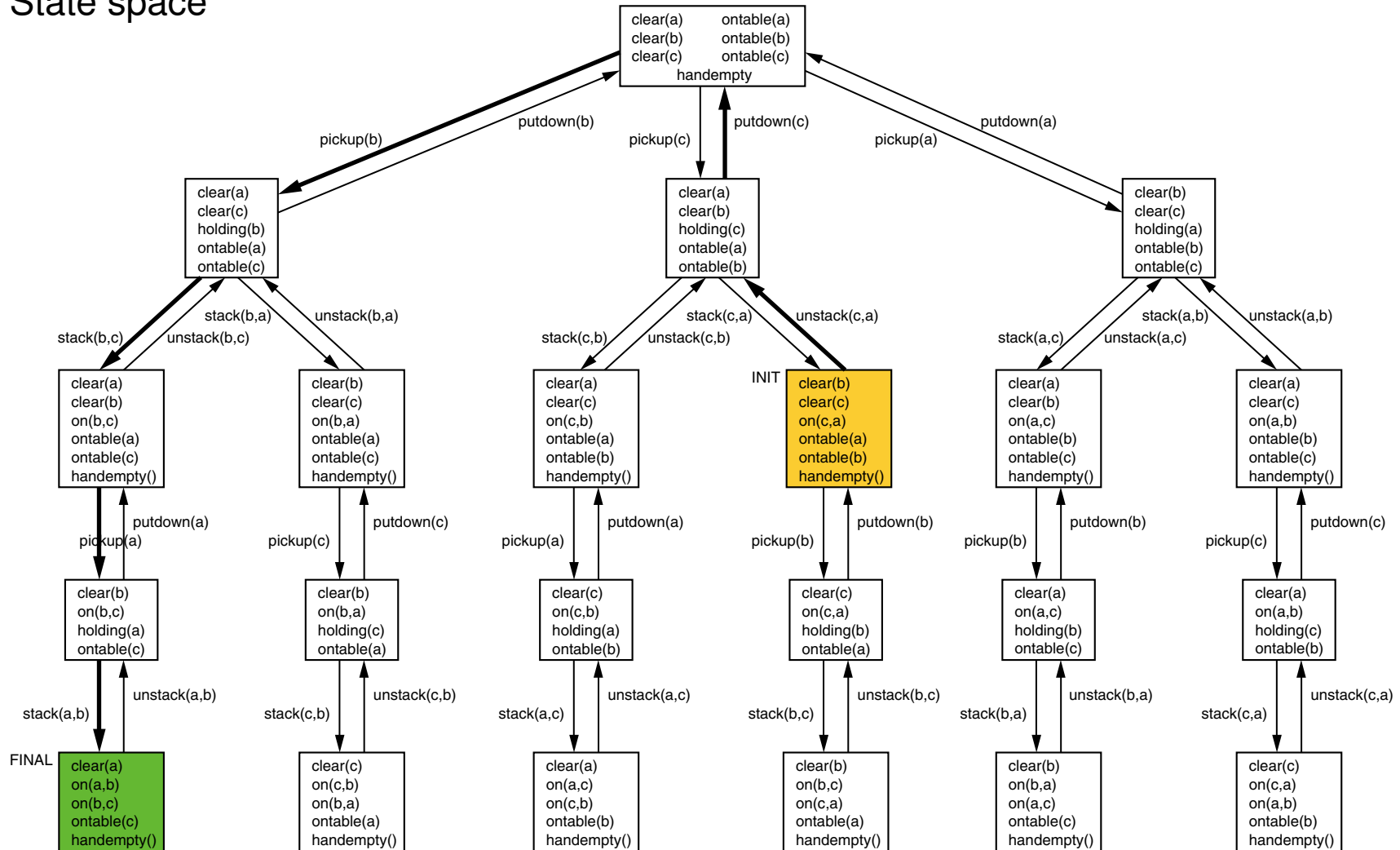
State-Space Planning
Heuristics by Simplified Models

Simplified models for a problem domain can be constructed by:

- ❑ Constraint Relaxation

  Relaxation of a model is the removal of constraints that prohibit the use of operators.

  ➜ Cost of an optimum solution for the simplified problem can be used as an estimate of optimum remaining cost.

- ❑ Overconstraining

  Additional constraints, e.g. fixing the initial part of solution paths, result in less complex models.

  ➜ Solution cost for such a model can be used as upper bounds for C*. (Pruning)

Both types of simplifications should result in models for which the problem can be solved easily. However, this effect cannot be guaranteed.

State-Space Planning

Example: Model Simplification for the 8-Puzzle Problem

STRIPS specification:

- ❏ Constants: $t_1, \ldots, t_8$ for tiles and $c_1, \ldots, c_9$ for cells.

5	2	3
8	■	4
7	1	6

- ❏ Predicates:

    $on(x, y)$ denotes tile x is in cell y.
    $clear(y)$ denotes cell y is empty.
    $adjacent(y, z)$ denotes cell z is adjacent to cell y.

- ❏ Initial state:

    $\{adjacent(c_1, c_2), adjacent(c_2, c_3), \ldots, on(t_5, c_1), on(t_2, c_2), \ldots, clear(c_5)\}$

- ❏ Operator:

operator :	$move(x, y, z)$:	(move tile x from cell y to cell z)
precond :	$on(x, y), clear(z), adjacent(y, z)$	
effects :	$on(x, z), clear(y), \neg on(x, y), \neg clear(z)$	

State-Space Planning

Example: Model Simplification for the 8-Puzzle Problem (continued)

Admissible heuristic functions:

h_1 number of misplaced tiles,

h_2 sum of Manhattan distances of misplaced tiles.

Q. Why is the admissibility of h_1 and h_2 so easy to see?

- ❏ Property $h(n) \leq h^*(n)$ has to be checked for all board configurations n of the 8-puzzle.
- ❏ $h^*(n)$ is unknown.

State-Space Planning

Example: Model Simplification for the 8-Puzzle Problem (continued)

Admissible heuristic functions:

h_1 number of misplaced tiles,

h_2 sum of Manhattan distances of misplaced tiles.

Q. Why is the admissibility of h_1 and h_2 so easy to see?

❑ Property $h(n) \leq h^*(n)$ has to be checked for all board configurations n of the 8-puzzle.

❑ $h^*(n)$ is unknown.

Simplification of rules:

❑ Simplification 1:
   Ignore that cells may be occupied. ➜ h_2

❑ Simplification 2:
   Ignore that cells may be occupied and allow arbitrary moves. ➜ h_1

State-Space Planning
Example: Model Simplification for the 8-Puzzle Problem (continued)

Relaxation of operator preconditions by deleting atoms:

1. Deletion of *clear*(z), *adjacent*(y, z).
   A tile can be placed directly on its target position, whether free or not.

   ➜ Heuristic function h_1 (number of misplaced tiles)

2. Deletion of *clear*(z).
   A tile can move to its target position without having cells to be empty in between.

   ➜ Heuristic function h_2 (sum of Manhattan distances)

3. Deletion of *adjacent*(y, z).
   A tile can be placed directly on the empty cell (Swap-Sort).

   ➜ New heuristic function h_3 ($\leq 1.5 \times$ number of misplaced tiles)

Relaxation of operator preconditions by adding atoms to states:

4. Allowing diagonal moves (additional *adjacent*(y, z) predicates in states).

   ➜ Searching for optimum solution paths in the relaxed problem is not trivial.

Remarks:

❑ A worst case scenario for the swap-sort approach is that we have only pairs of tiles that need to swap positions. Each swap takes three moves.

❑ Additional deletions can be considered when reformulating the 8-puzzle problem, e.g. by

$$adjacent(x, y) \Leftrightarrow neighbor(x, y) \wedge (\textbf{\textit{same\_row}}(x, y) \vee \textbf{\textit{same\_column}}(x, y))$$

State-Space Planning

Example: Model Simplification for the 8-Puzzle Problem (continued)

- ❑ Trivial observation:

  Optimum solution cost for relaxed problems is an admissible estimate for optimum solution cost in the 8-puzzle model.

  Justification:

  Applicability of operators is increased in the relaxed models. All action sequences for the 8-puzzle model will also be considered for the relaxed models. Therefore, optimum solution cost in relaxed models is not greater than optimum solution cost in the original model.

- ❑ Not so trivial observation:

  Optimum solution cost for relaxed problems is a monotone estimate for optimum solution cost in the 8-puzzle model.

  Justification:

  Optimum solution cost is no estimate in the relaxed problems.

- ➜ Real question:

  When is an optimum solution efficiently computable in a relaxed model?

State-Space Planning

Generation of Heuristics by Simplified Models

Criteria for the recognition of easy problems

❑ **Decomposability of subgoals**

Subgoals can be solved independently, i.e. each subgoal can be satisfied by some actions without undoing the effect of previous actions and without affecting the applicability of future actions.

(See relaxation approach 1 for the 8-puzzle problem.)

❑ **Partial order of subgoals**

Subgoals can be ordered in such a way that the solution of a subgoal no longer affects a subgoal that has already been solved.

(See relaxation approach 3 for the 8-puzzle problem.)

❑ **Commutativity of operators**

The internal order at which a given set of operators is applied does not alter the set of operators applicable in the future.

(The greedy algorithm "cheapest-subgoal-first" can be used to achieve an optimal solution.)

State-Space Planning

Relaxation of operator preconditions by deleting atoms:

1. Deletion of *clear*(z), *adjacent*(y, z): Decomposability

   A conjunction of predicates *on*(X_i, C_j) defines the goal state. When using this relaxation, each target condition *on*(X_i, C_j) represents an independently solvable subproblem – immediate placement of the tile in its target cell.

2. Deletion of *clear*(z): Decomposability

   Each target condition *on*(X_i, C_j) represents an independently solvable subproblem. Action sequences for subgoals can be executed sequentially.

3. Deletion of *adjacent*(y, z): Partial Order

   Subgoals and their associated operators can be ordered such that the operators designated for any subgoal may influence only subgoals of higher order, leaving all other subgoals unaffected. Establishing the correct position of the empty cell is a subgoal of a higher order than all the other subgoals and should, therefore, be attempted last.

State-Space Planning

Backward Search in State Spaces (Regression Planning)

❑ Instead of starting with the initial state, we could start with the goal description in order to add only "necessary" actions to a plan.

State-Space Planning

Backward Search in State Spaces (Regression Planning)

❑ Instead of starting with the initial state, we could start with the goal description in order to add only "necessary" actions to a plan.

❑ Backward search does not work with states (sets of positive literals only) like forward search but with goals (sets of positive and negative literals). Goals (no CWA) can be understood as partially defined states (with CWA).

State-Space Planning

Backward Search in State Spaces (Regression Planning)

❑ Instead of starting with the initial state, we could start with the goal description in order to add only "necessary" actions to a plan.

❑ Backward search does not work with states (sets of positive literals only) like forward search but with goals (sets of positive and negative literals). Goals (no CWA) can be understood as partially defined states (with CWA).

❑ The starting point of backward search is the inverse of the state transition relation (weakest precondition):

$g' \xrightarrow{a}{}^{-1} g$ holds   iff g is a minimal goal such that
$\quad\quad\quad\quad\quad\quad\quad\quad$ a is applicable for all states s satisfying g and
$\quad\quad\quad\quad\quad\quad\quad\quad$ the corresponding successor states s' satisfy g'.

The search space graph is described by the goal *goal* as start node and the inverse transition relation T^{-1}.

❑ As in the case of forward search, deterministic implementations can be specified for the search in this search space graph.

Remarks:

- ❏ When a goal is called minimal in the sense of "Weakest Preconditions" has been left open here. In the presented simple case of regression planning it is possible to use as minimization criterion the number of positive and negative literals contained in the goal.

- ❏ While forward search could insert actions into the plan that are not useful, backward search actions appear to be "necessary" for reaching the goal.

State-Space Planning

Regression Planning To create a plan for a list of goal literals, we try to achieve all the literals simultaneously, i.e., beginning a plan from the end.

Nondeterministic algorithm:

1. Start with an empty plan p from the given goal g.

2. While the current goal conditions are not all satisfied in the initial state s_{init} do:

    (a) Choose a subgoal.   // Even a subgoal satisfied in s_{init} can be chosen.

    (b) If no action can achieve this subgoal, then return *Failure*.

    (c) Choose an action that achieves this subgoal (i.e. an operator with a ground instantiation that forms an action achieving the subgoal).

    (d) Insert this action at the beginning of the plan p.

    (e) The new goal is a weakest precondition of the current goal, i.e. a minimum set of facts describing states in which the selected action is executable, so that **all** current goal conditions are satisfied in the successor state.

3. Return the plan p.

Remarks:

- ❏ When assuming that subgoals can be solved independently, we should distinguish between "selection" (of an unfulfilled subgoal) and "choice" (of an action). In this case, it is only a strategic decision which subgoal is to be used next, all subgoals must be met anyway. Different choices for an action, however, may lead to different results e.g. success or failure. Therefore, nondeterminism is in the choices, not in selections.

State-Space Planning

Example: Regression Planning in Blocks World

- ❏ Constants: a, b, c, d

- ❏ Predicates: $on(x, y)$, $ontable(x)$, $holding(x)$, $clear(x)$, $handempty()$

- ❏ Operators:

  . . .

  operator: $unstack(x, y)$
      precond: $on(x, y)$, $clear(x)$, $handempty()$
      effects: $holding(x)$, $clear(y)$, $\neg on(x, y)$, $\neg clear(x)$, $\neg handempty()$

- ❏ Goal: $\{holding(b), clear(c)\}$

  Operator $unstack(x, y)$ can achieve subgoal $holding(b)$.

- ❏ Determining Weakest Preconditions:

  1. Substitution: $[x/b, y/c]$ Action: $unstack(b, c)$

     New goal: $\{on(b, c), clear(b), handempty()\}$

  2. Substitution: $[x/b, y/a]$ Action: $unstack(b, a)$

     New goal: $\{on(b, a), clear(b), clear(c), handempty()\}$

  3. Substitution: $[x/b, y/d]$ Action: $unstack(b, d)$

     New goal: $\{on(b, d), clear(b), clear(c), handempty()\}$

State-Space Planning

Determining Weakest Preconditions

Definition 7 (Regression of a Literal, Weakest Precondition)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given. Let g be a goal description in $\mathcal{S}$, i.e. a finite set of variable-free literals. Further, let a be an action, i.e. a ground instance of an operator in O.

The regression *regression*(l, a) of a literal $l \in g$ for a is defined as

$$regression(l, a) = \begin{cases} \textit{true} & l \in \text{effects}(a) \\ \textit{false} & \neg l \in \text{effects}(a) \\ l & \text{otherwise} \end{cases}$$

The *weakest precondition wp*(g, a) of g for a is defined as

$$wp(g, a) = \{regression(l, a) \mid l \in g\} \cup \text{precond}(a)$$

If an atomic formula occurs positively and negatively in the effects of an action, we just consider the positive occurrence (following the STRIPS commitment "First delete, then add." in action application). For simplicity this is no incorporated in the definition.

Remarks:

- ❏ For simplicity, the positive literal is often referred to as $\neg l$ for a negated literal l.

- ❏ A literal *true* in a Weakest Precondition is satisfied in every state. We can omit it.

- ❏ A literal *false* in a Weakest Precondition cannot be satisfied in any state. Therefore, a weakest precondition containing *false* can be seen as a dead end. Search can be aborted for the current plan.

- ❏ When applying an action to a legal state, the successor state should be legal as well (assuming an appropriate STRIPS model). However, when determining the weakest precondition of an action, inconsistent partial state descriptions may occur.

- ❏ If an action is considered that contains an atomic formula negated and unnegated as effects, the computation of weakest preconditions is more involved. The definition of *regression*(l, a) must take into account the order of effects in the computation of successor states: first delete literals, then add literals.

$$
regression(l, a) = \begin{cases}
\textit{true} & l \text{ positive and } l \in \text{effects}(a) \\
\textit{true} & l \text{ negative and } l \in \text{effects}(a) \text{ and } \neg l \notin \text{effects}(a) \\
\textit{false} & l \text{ positive and } \neg l \in \text{effects}(a) \text{ and } l \notin \text{effects}(a) \\
\textit{false} & l \text{ negative and } \neg l \in \text{effects}(a) \\
l & \text{otherwise}
\end{cases}
$$

State-Space Planning

Example: Regression Planning in Blocks World (continued)

- ❑ Constants: a, b, c, d

- ❑ Predicates: *on*(x, y), *ontable*(x), *holding*(x), *clear*(x), *handempty*$()$

- ❑ Operators:

  . . .

  operator: *stack*(x, y)

    precond: *holding*(x), *clear*(y)

    effects: *on*(x, y), *clear*(x), ¬*clear*(y), ¬*holding*(x), *handempty*$()$

- ❑ Goal: $\{$*on*(b, c), *on*$(a, b)\}$

  Operator *stack*(x, y) can achieve subgoal *on*(b, c).

- ❑ Determining the "Weakest Precondition":

  Substitution: $[x/b, y/c]$

  Action: *stack*(b, c)

  Weakest Precondition: $\{$*holding*(b), *clear*(c), *on*$(a, b)\}$

➜   Weakest precondition cannot be satisfied: semantic inconsistency!

State-Space Planning

Example: Regression Planning in Blocks World (continued)

- ❑ Constants: a, b, c, d

- ❑ Predicates: $on(x, y), ontable(x), holding(x), clear(x), handempty()$

- ❑ Operators:

  . . .

operator:	$pickup(x)$
precond:	$ontable(x), clear(x), handempty()$
effects:	$holding(x), \neg ontable(x), \neg clear(x), \neg handempty()$

operator:	$putdown(x)$
precond:	$holding(x)$
effects:	$ontable(x), clear(x), handempty(), \neg holding(x)$

- ❑ Goal: $\{holding(a), clear(b), on(b, c)\}$

- ❑ Action $pickup(x)$ achieves subgoal $holding(a)$ by substitution $[x/a]$

  Weakest Precondition: $\{ontable(a), clear(a), handempty(), clear(b), on(b, c)\}$

- ❑ Action $putdown(x)$ achieves subgoal $clear(a)$ by substitution $[x/a]$

  Weakest Precondition: $\{holding(a), clear(b), on(b, c)\}$

➜ Cycle in the sequence of weakest preconditions!

Remarks:

- ❑ Semantic inconsistencies typically result from incomplete or inadequate modeling. Syntactic inconsistencies are easily recognized: the literal "*false*" or the same literal occurring negatively and positively in a weakest precondition indicate a dead end.

- ❑ To detect cycles, it is not enough to determine whether goals occur more than once. Since goals are only partially defined states, the backward search can generate supersets of previous goals. However, supersets of a goal do not represent progress.

- ❑ In a backward search avoiding nonsense actions such as *unstack*(a, a) is enormously important. By inserting additional constraints in the precond part, in the example $x \neq y$, undesired specializations can be avoided. A consistency test of all variable-free (in-)equations is easy because of the Unique Name Assumption.

- ❑ Please note that $x \neq y$ is a literal with predefined semantics. Using equality in STRIPS as predicate symbol therefore leads to a more complex planning language. In the context of regression planning without lifting, however, we can avoid the equality symbol by adding preconditions $\neg equal(x, y)$ to operator descriptions and literals $equal(c, c)$ for all constants c to state descriptions.

State-Space Planning

Regression Planning (continued)

❑ Plans from Regression Planning

A plan is returned in Regression Planning if the current goal is satisfied in the initial state. By definition of weakest preconditions and by construction of the plan, the action sequence is applicable to the initial state.

❑ Regression Planning is *sound*.

If a plan is returned, this plan solves the planning problem at hand.

❑ Regression Planning is *complete*.

If a plan exists that is a solution to the planning problem, then such a plan can be minimized by eliminating actions that are not needed to reach the goal. A minimal plan can be found by Regression Planning.

State-Space Planning

Example: Regression Planning in Blocks World (continued)

- ❏ Constants: a, b, c, d

- ❏ Predicates: *on*(x, y), *ontable*(x), *holding*(x), *clear*(x), *handempty*$()$

- ❏ Operators:

  . . .

operator:	*unstack*(x, y)
precond:	*on*(x, y), *clear*(x), *handempty*$()$
effects:	*holding*(x), *clear*(y), $\neg$*on*(x, y), $\neg$*clear*(x), $\neg$*handempty*$()$

- ❏ Goal: $\{$*holding*(b), *clear*$(c)\}$

- ❏ Operator: *unstack*(b, y)
  Necessary Substitution: $[x/b]$ (Least Commitment)

  Determining Weakest Preconditions:

  New Goal: $\{$*on*(b, c), *clear*(b), *handempty*$()\}$
  or
  New Goal: $\{$*on*(b, y), $y \neq c$, *clear*(b), *clear*(c), *handempty*$()\}$

➜ Lifting (= Partial Instantiation, Lifting to First Order Logic)

State-Space Planning

Lifted Regression Planning in State Spaces

❑ When using lifting, goal descriptions must be allowed to contain variables.

❑ A new set of variables has to be used for an operator each time it is applied for generating a subgoal.

❑ In order to avoid redundancies (overlapping of different goal descriptions), inequalities for variables and constants can occur in goal descriptions.

Definition 8 (Lifted STRIPS Goal)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given.

A *lifted STRIPS goal* g is described by a finite set (i.e. conjunction) of literals and a finite set of inequalities.

A lifted goal g is satisfied (or achieved) modulo some substitution σ in a state s if applying the substitution σ to g results in a goal (i.e. variable-free set of literals) that has only true equalities w.r.t. UNA and DC and for which all other literals are satisfied in s.

A lifted plan is a list of partially instantiated operators.

Remarks:

- ❑ If a lifted goal is satisfied modulo some substitution, the substitution has to replace all variables in the goal by constants.

State-Space Planning

Lifted Regression Planning in State Spaces (continued)

Instead of looking at ground instances for operators, as with backward searches, the operators are specialized by substitutions only to the extent necessary to reach a goal.

Nondeterministic algorithm:

1. Start with an empty plan p and an empty substitution σ from the given goal.

2. While there is no extension of σ such that all the current goal conditions are satisfied in the initial state s_{init} do:

    (a) Choose a subgoal.   // A subgoal satisfied modulo substitution in s_{init} can be chosen.

    (b) If no action can achieve this subgoal, then return *Failure*.

    (c) Choose an instantiation of an operator (usually not ground, but with new variable names) that achieves this subgoal, extending σ when needed.

    (d) Insert this partially instantiated operator at the beginning of the plan p.

    (e) The new goal is a weakest precondition of the current goal,
         i.e. a minimum set of facts describing states in which the selected partially instantiated operator is "executable", so that **all** current goal conditions are satisfied in the successor state.

3. Return the plan p substituted by an extension σ' of σ such that $\sigma'(p)$ is variable-free and the current goal modulo σ' is satisfied in s_{init}.

Remarks:

❑ After termination of the above algorithm, not all variables occurring in the lifted plan may have been substituted by constants via σ. Here, some arbitrary replacement can be done that is meaningful at time of execution. But usually, this is an indicator of bad modeling or of some strange domain.

State-Space Planning
Determining Lifted Weakest Preconditions

Definition 9 (Lifted Regression of a Literal, Lifted Weakest Precondition)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given. Let g be a goal description in $\mathcal{S}$ for lifting, i.e. a finite set of literals an inequalities. Let o be an operator with variables renamed to $o(x_1, ..., x_m)$ where $x_1, ..., x_m$ are new variables. Let σ be some substitution of variables in g and $o(x_1, ..., x_m)$.

Further,

❑ let $\sigma_1, \ldots, \sigma_u$ be the most general unifiers such that l is achieved,
i.e. $\sigma_i(\sigma(l)) \in$ *effects*$(\sigma_i(\sigma(o)))$.

❑ let $\sigma_{u+1}, \ldots, \sigma_{u+v}$ be the most general unifiers such that l is contradicted,
i.e. $\sigma_i(\sigma(\neg l)) \in$ *effects*$(\sigma_i(\sigma(o)))$.

The regression *regression*$(l, o(x_1, ..., x_m), \sigma' \circ \sigma)$ of a literal $l \in g$ for o w.r.t. $\sigma' \circ \sigma$ is defined as

regression$(l, o(x_1, ..., x_m), \sigma' \circ \sigma)$

$$
= \begin{cases}
\textit{true} & \text{(a)} \quad \sigma' = \sigma_i \text{ for some } i = 1, \ldots, u \\
\textit{false} & \text{(b)} \quad \sigma' = \sigma_j \text{ for some } j = u + 1, \ldots, u + v \\
\sigma(l), y_1 \neq \sigma_1(y_1), \ldots, y_{u+v} \neq \sigma(y_{u+v}) & \text{(c)} \quad \sigma' = [\,] \text{ and there is a choice of} (y_1, \ldots, y_{u+v}) \\
& \qquad \text{such that } y_1 \neq \sigma_1(y_1), \ldots, y_{u+v} \neq \sigma_{u+v}(y_{u+v})
\end{cases}
$$

The *lifted weakest precondition* $wp(g, a, \sigma_f)$ of g for o is made up of the preconditions of $\sigma_f(o(x_1, \ldots, x_m))$ and the result of iteratively applying the above single step definition for g subgoal by subgoal. In the first step an empty substitution is used as σ. Then, the resulting substitution of the previous step is used as σ in the next step, finally resulting in σ_f.

Remarks:

- ❑ An MGU (most general unifier) σ' is a substitution that unifies two literals l and l' (i.e. $\sigma'(l) = \sigma'(l')$) by substituting a minimum number of variables by terms. A MGU is unique (up to renaming of variables) if the literals can be unified.

- ❑ Note that each operation is applied with new variable names that have not been used before (i.e. instead of $o(x_1, ..., x_m)$ from the STRIPS specification use $o(x'_1, ..., x'_m)$ with variables $x'_1, ..., x'_m$ that have not been used so far). Otherwise, the specific MGUs would be too special.

- ❑ For the selection of variables for the prohibition of certain MGUs (selection of $(y_1, \ldots, y_{u+v})$ in definition 9) there are only finitely many possibilities, because only finitely many variables can be changed in a MGU of two formulas. Selecting an unmodified variable results in a $x \neq x$ condition that can never be met, so such cases don't have to be considered.

- ❑ Definition 9 describes the different choices for substitutions per literal. By regressing all literals of the current goal one by one, a final substitution is constructed.

- ❑ The determination of regression by means of an operation with lifting allows different alternatives. All these alternatives have to be considered successively by a deterministic procedure.

State-Space Planning

Determining Lifted Weakest Preconditions (continued)

- ❑ The operation o and an initial substitution σ results from the selection of a literal from the current target s, which can be reached with the operation o using σ.

- ❑ The various alternative weakest preconditions are obtained by applying one of the regressions (different choices for σ' resp. for $(y_1, \ldots, y_{u+v})$ in definition 9) for each additional literal of the current target and adding the preconditions o with the final substitution.

➜ The number of alternative regressions is related to the number of literals in effects(o). Different plans result from different operator selections and instantiations.

➜ Since goals can contain variables during regression planning with lifting, they combine a large number of possible goals that would result from using ground instances of operators. The branching degree of the search space is therefore lower with Lifting.

State-Space Planning

Example 1: Lifted Weakest Preconditions in Blocks World

- ❏ Current goal: $\{clear(c), holding(b)\}$

- ❏ Selected subgoal: $holding(b)$

- ❏ Selected operator: $unstack(x, y)$ with substitution $\sigma = [x/b]$

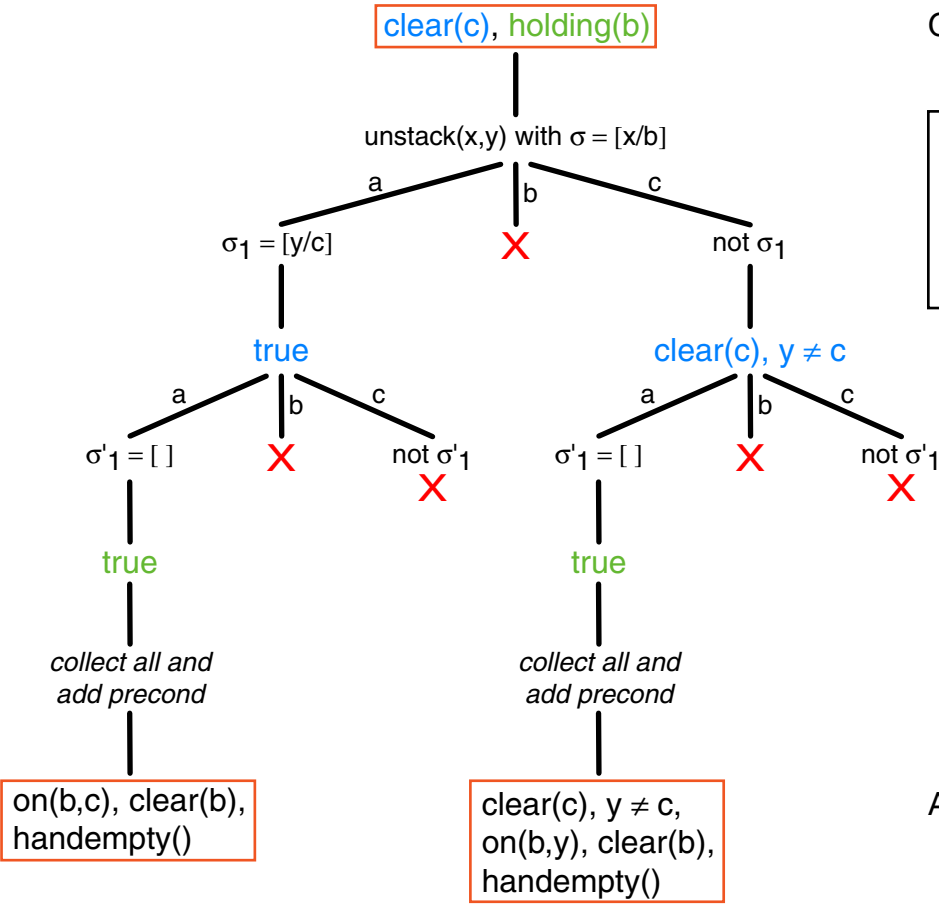 | operator: | $unstack(x, y)$ |
  |---|---|
  | *precond*: | $on(x, y), clear(x), handempty()$ |
  | effects: | $holding(x), clear(y), \neg on(x, y), \neg clear(x), \neg handempty()$ |

- ❏ Regression of $clear(c)$ using $unstack(x, y)$ and $\sigma = [x/b]$

  Case (a): An extension of the substitution by $[y/c]$ returns this goal formula as an effect. The resulting substitution then is $[x/b, y/c]$. Case (b): $\neg clear(.)$ is not contained in effects($unstack(x, y)$). So, this case cannot occur.
  Case (c): Since there is only one substitution for case (a), $clear(c)$ and $y \neq c$ have to be added to the weakest precondition.

State-Space Planning

Example 1: Lifted Weakest Preconditions in Blocks World (continued)

clear(c), holding(b)   Goal

unstack(x,y) with σ = [x/b]

a   b   c

σ₁ = [y/c]   X   not σ₁

true   clear(c), y ≠ c

a   b   c   a   b   c

σ'₁ = [ ]   X   not σ'₁   σ'₁ = [ ]   X   not σ'₁
              X   X

true   true

collect all and   collect all and
add precond   add precond

on(b,c), clear(b),   clear(c), y ≠ c,
handempty()   on(b,y), clear(b),
                        handempty()

holding(b)	produced with σ = [x/b] by
operator:	unstack(x,y)
precond :	on(x,y), clear(x), handempty()
effects :	holding(x), clear(y), ¬on(x,y), ¬clear(x), ¬handempty()

Alternative Weakest Preconditions

State-Space Planning

Comparison with regression without lifting for *unstack*(x, y):
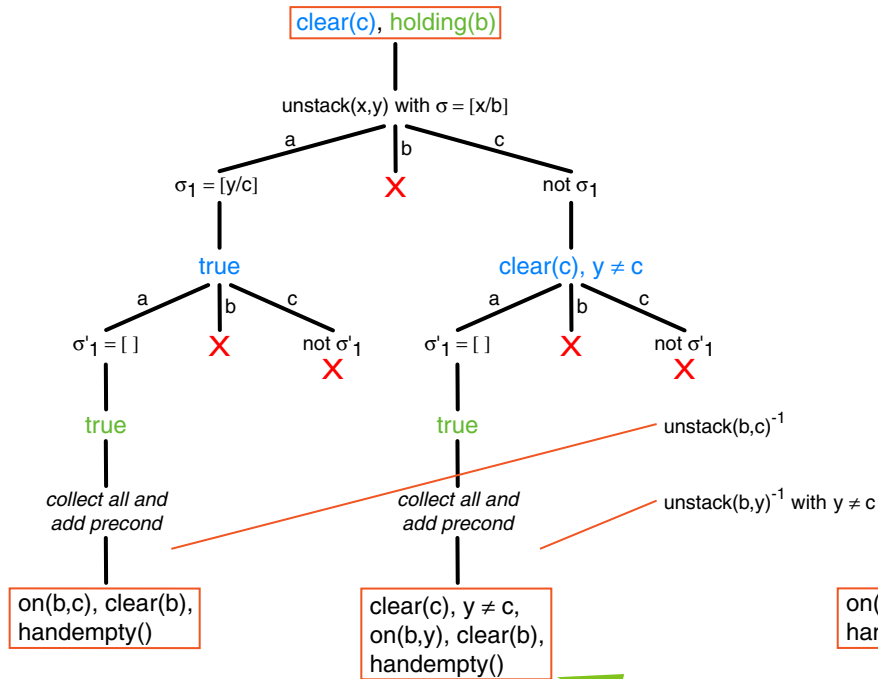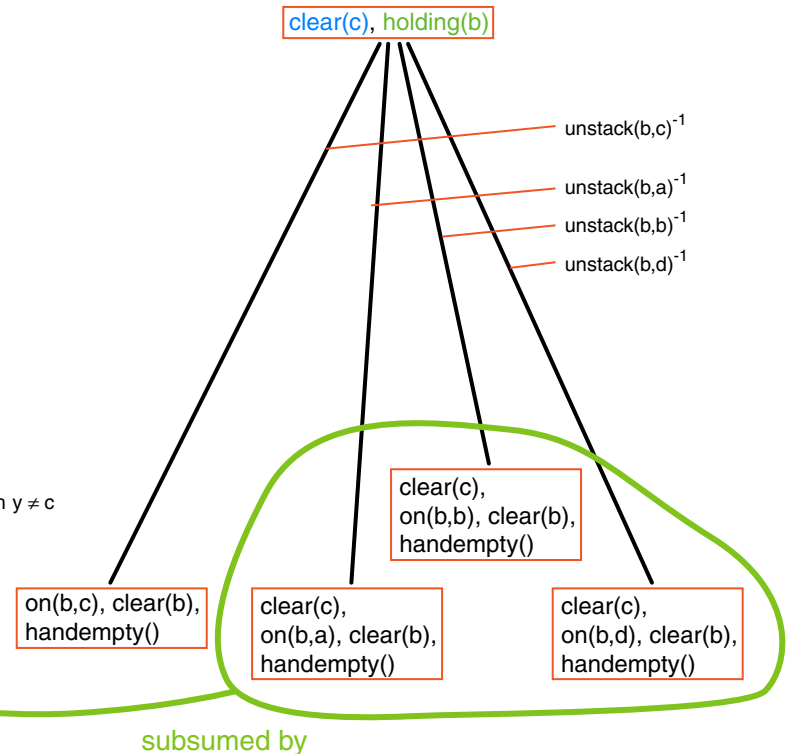
State-Space Planning

Example 2: Lifted Weakest Preconditions in Blocks World

❏ Current goal: $\{holding(b), ontable(c), clear(a), \neg clear(z)\}$

❏ Selected subgoal: $holding(b)$

❏ Selected operator: $unstack(x, y)$ with substitution $\sigma = [x/b]$

> operator: $unstack(x, y)$
>   precond: $on(x, y), clear(x), handempty()$
>   effects: $holding(x), clear(y), \neg on(x, y), \neg clear(x), \neg handempty()$

❏ Regression of $ontable(c)$ using $unstack(x, y)$ and $\sigma = [x/b]$

   Case (a), (b): Predicate $ontable$ does not occur in effects($unstack(x, y)$). So, for this literal we only have case (c) for a regression.
   Case (c): Since there are no substitutions for case (a) or (b), only $ontable(c)$, but no inequality is added to the weakest precondition.

❏ Regression of $clear(a)$ using $unstack(x, y)$ and $\sigma = [x/b]$

   Case (a): An extension of the substitution by $[y/a]$ returns this goal formula as an effect. The resulting substitution then is $[x/b, y/a]$. Case (b): No substitution can provide the negated goal formula. Case (c): The weakest precondition has to be be supplemented by $clear(a)$ and $y \neq a$.

State-Space Planning

Example 2: Lifted Weakest Preconditions in Blocks World   (continued)

❏ Current goal: $\{holding(b), ontable(c), clear(a), \neg clear(z)\}$

❏ Selected subgoal: $holding(b)$

❏ Selected operator: $unstack(x, y)$ with substitution $\sigma = [x/b]$

    operator: $unstack(x, y)$
       precond: $on(x, y), clear(x), handempty()$
      effects: $holding(x), clear(y), \neg on(x, y), \neg clear(x), \neg handempty()$

❏ Regression of $\neg clear(z)$ by $unstack(x, y)$ and $\sigma = [x/b]$

Case (a): An extension of the substitution by $[z/b]$ returns the goal formula as an effect.
Case (b): An extension of the substitution by $[z/y]$ returns the negated goal formula as an effect.
Case (c): As a consequence, we have to add $\neg clear(z)$, $z \neq y$, and $z \neq b$ to the weakest precondition.
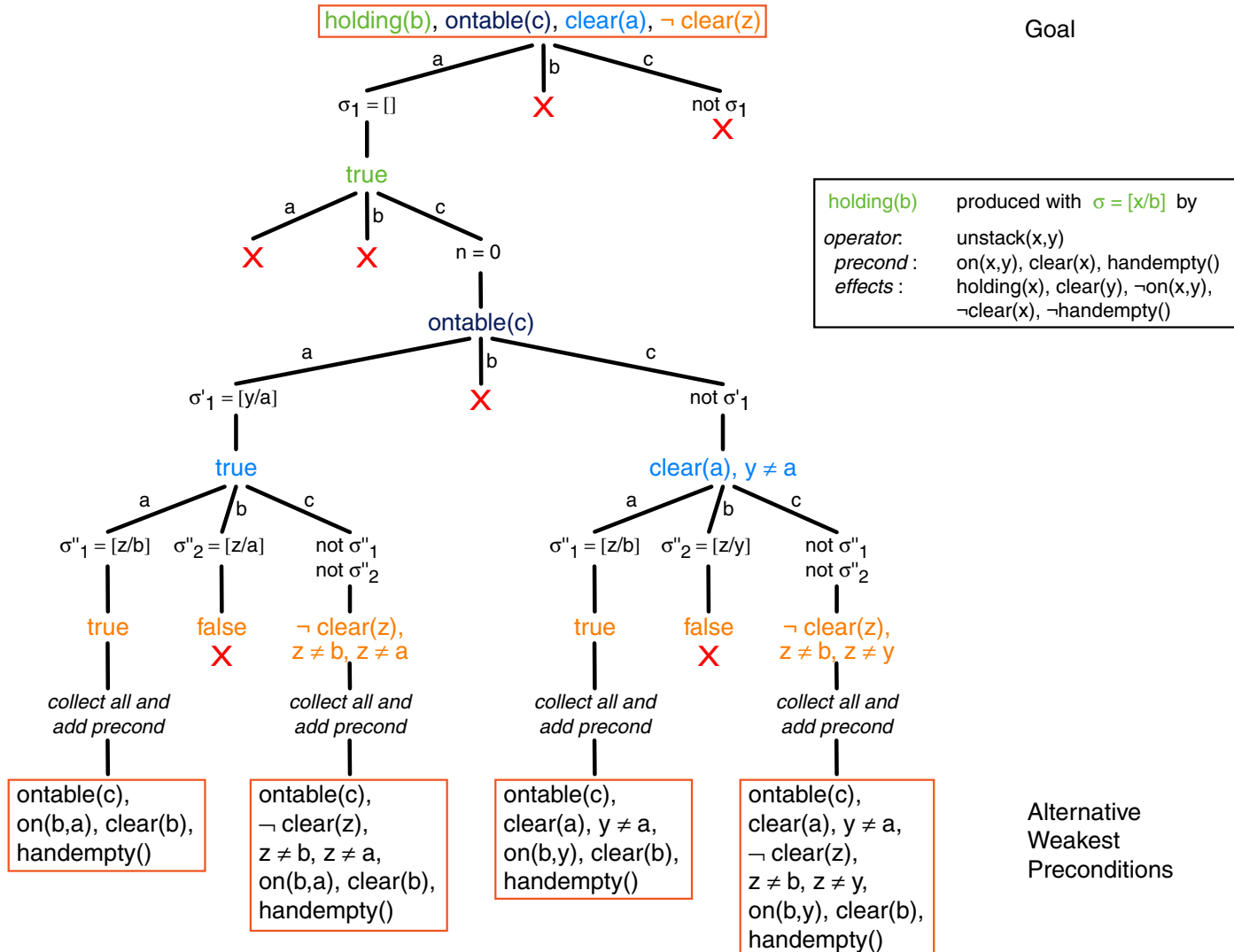
❏ Regression of $unstack(x, y)$ and $\sigma = [x/b, z/b]$

The resulting weakest precondition is in this case

$$\{ontable(c), clear(a), y \neq a, on(b, y), clear(b), handempty()\}$$

(y can be seen as existentially quantified variable when looking for states satisfying this weakest precondition.)

State-Space Planning

Example 2: Lifted Weakest Preconditions in Blocks World (continued)

holding(b), ontable(c), clear(a), ¬ clear(z)   Goal

holding(b)	produced with	σ = [x/b] by
operator:	unstack(x,y)	
precond :	on(x,y), clear(x), handempty()	
effects :	holding(x), clear(y), ¬on(x,y), ¬clear(x), ¬handempty()	

σ₁ = [] → true

n = 0 → ontable(c)

σ'₁ = [y/a] → true

clear(a), y ≠ a

σ''₁ = [z/b] σ''₂ = [z/a] not σ''₁ not σ''₂

true   false   ¬ clear(z), z ≠ b, z ≠ a

σ''₁ = [z/b] σ''₂ = [z/y] not σ''₁ not σ''₂

true   false   ¬ clear(z), z ≠ b, z ≠ y

collect all and add precond

```
ontable(c),
on(b,a), clear(b),
handempty()
```

```
ontable(c),
¬ clear(z),
z ≠ b, z ≠ a,
on(b,a), clear(b),
handempty()
```

```
ontable(c),
clear(a), y ≠ a,
on(b,y), clear(b),
handempty()
```

```
ontable(c),
clear(a), y ≠ a,
¬ clear(z),
z ≠ b, z ≠ y,
on(b,y), clear(b),
handempty()
```

Alternative Weakest Preconditions

Remarks:

❑ It should be noted that substitutions for variables that already occur in the current goal have an effect on the task list search carried out so far. The replacements must also be carried out there, particularly in the current plan.

State-Space Planning
Recursive STRIPS-Planning

Idea: To create a plan for a list of goals, select a sub-goal, create a plan to achieve this goal, append a plan to it to achieve the remaining goals.

Nondeterministic algorithm:

1. Start with an empty plan p from the initial state s_{init}.

2. While the current state does not satisfy the goal conditions g do:

    (a) Choose a subgoal that is not satisfied in the current state.

    (b) If no action can achieve this subgoal, then return *Failure*.

    (c) Choose an action that can achieve the subgoal.

    (d) Recursion: Determine a subplan that starts from the current state and has the preconditions of the selected action as goal.

    (e) If no subplan was found, return *Failure*.

    (f) Append the selected action to the subplan.

    (g) Determine the successor state of the subplan and use it as new current state.

    (h) Append the subplan to the plan p.

3. Return the plan p.

Remarks:

❏ Recursive STRIPS planning uses a "divide and conquer" strategy.

❏ As there is no guarantee that subgoals con be achieved independently, plans that achieve different subgoals can interfere with each other. In Recursive STRIPS Planning subgoals are reestablished if they are destroyed by subsequent subplans. In such cases recursive STRIPS planning will not find shortest plans.

❏ In Nilsson's implementation of Recursive STRIPS Planning, there is no backtracking for parts of the search that have already become part of the plan.

State-Space Planning

Example: Sussman Anomaly in Blocks World
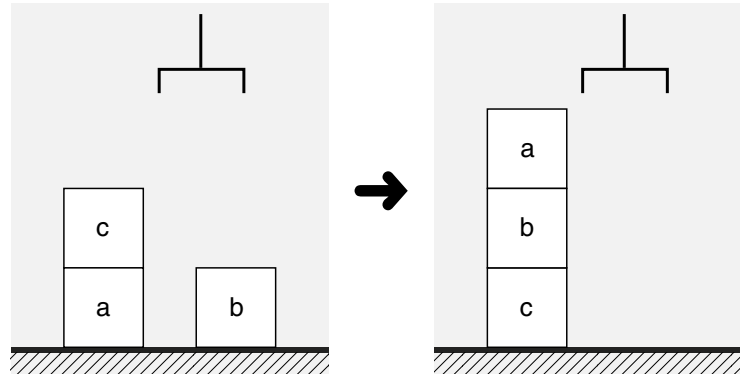
❑ Problem

❑ Initial state: $\{clear(b), clear(c), on(c, a), handempty(), ontable(a), ontable(b)\}$

❑ Goal: $\{on(b, c), on(a, b)\}$

State-Space Planning
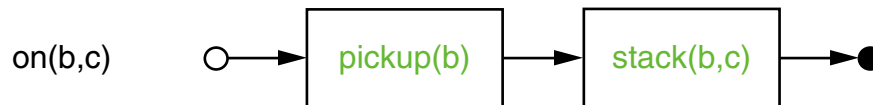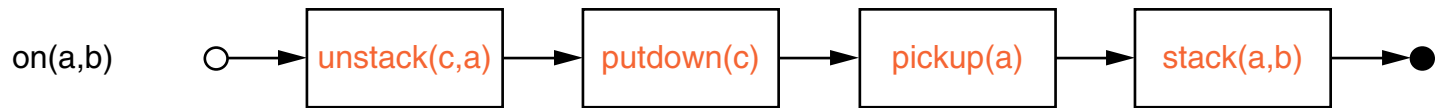
Example: Sussman Anomaly in Blocks World

❏ Problem
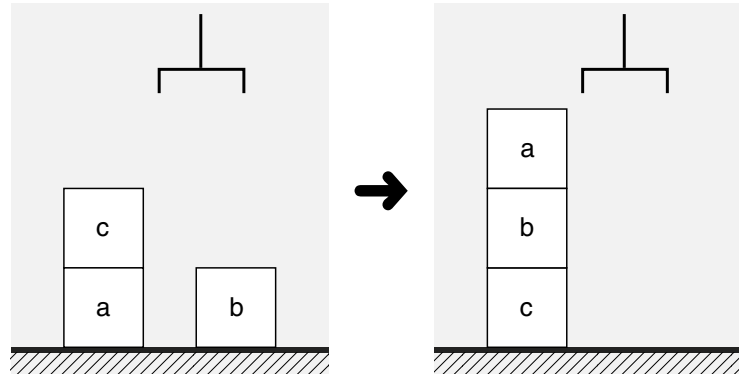
❏ Initial state: $\{clear(b), clear(c), on(c, a), handempty(), ontable(a), ontable(b)\}$

❏ Goal: $\{on(b, c), on(a, b)\}$

❏ Shortest plans for subgoal $on(b, c)$ resp. $on(a, b)$
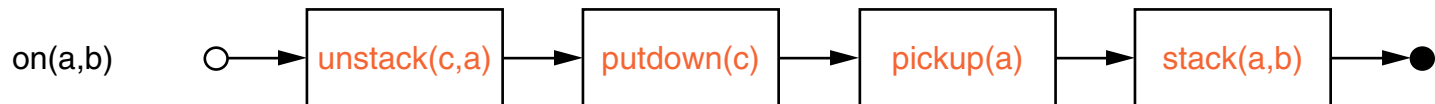
State-Space Planning

Example: Sussman Anomaly in Blocks World   (continued)

❑ Shortest plans for subgoal $on(b, c)$ resp. $on(a, b)$

on(b,c) ○ → [pickup(b)] → [stack(b,c)] → ●

on(a,b) ○ → [unstack(c,a)] → [putdown(c)] → [pickup(a)] → [stack(a,b)] → ●

❑ Recursive STRIPS Planning:
Subgoal $on(a, b)$ will be destroyed when considering $on(b, c)$ afterwards.
Subgoal $on(b, c)$ will be destroyed when considering $on(a, b)$ afterwards..

➜   Recursive STRIPS Planning will not always return shortest plans.

❑ Shortest plan for $\{on(b, c), on(a, b)\}$

○ → [unstack(c,a)] → [putdown(c)] → [pickup(b)] → [stack(b,c)] → [pickup(a)] → [stack(a,b)] → ●
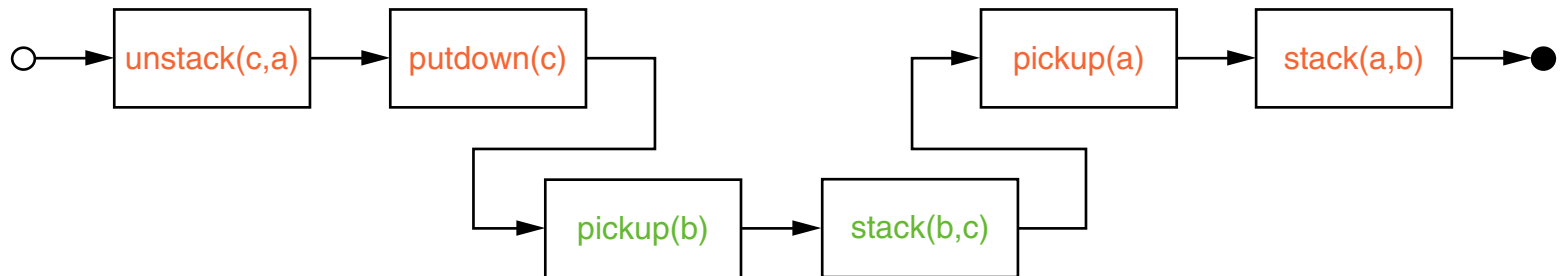
Remarks:

❑ Alternatively to reestablishing them, subgoals could be protected against deletion by future actions. But this approach can lead to incompleteness.

State-Space Planning

Example: Register Swapping Problem
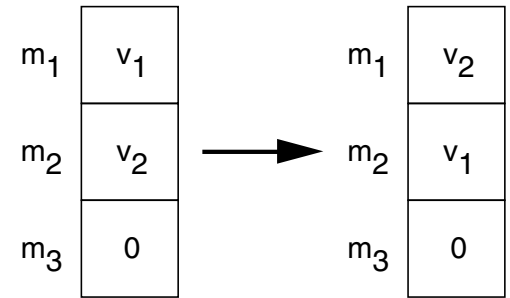
- □ Constants: $m_1, m_2, m_3, v_1, v_2, 0$

- □ Predicates: *contains*(x, y)

- □ Operators:

  operator: *assign*(x_1, y_1, x_2, y_2)

      precond: *contains*(x_1, y_1), *contains*(x_2, y_2)

      effects: $\neg$*contains*(x_1, y_1), *contains*(x_1, y_2)

m_1	v_1		m_1	v_2
m_2	v_2	→	m_2	v_1
m_3	0		m_3	0

- □ Initial state: $\{$*contains*(m_1, v_1), *contains*(m_2, v_2), *contains*$(m_3, 0)\}$

- □ Goal: $\{$*contains*(m_1, v_2), *contains*$(m_2, v_1)\}$

- □ Plan: (*assign*$(m_3, 0, m_2, v_2)$, *assign*(m_2, v_2, m_1, v_1), *assign*(m_1, v_1, m_3, v_2))

➜ Nilsson's deterministic version of Recursive STRIPS Planning (with Lifting) cannot find a plan!

Remarks:

❑ Nilsson's implementation of the Recursive STRIPS Planning algorithm uses lifting. Backtrack points in the procedure are the selection of a subgoal and the selection of the action to achieve this subgoal. When using lifting and selecting a subgoal containing variables, this subgoal may be found in the current state (loop condition) using an appropriate substitution. Even in such cases the algorithm will not try to achieve the subgoal with an operator when backtracking. (See also Nilsson: Principles of Artificial Intelligence, p. 298-307.)

Plan-Space Planning
Differences to State-Space Planning

❑ State-Space Planning

  – nodes = states /sets of states of the domain

  – edges = state transitions by actions

  – plan = path in the state space

❑ Plan-Space Planning

  – nodes = incomplete plans

  – edges = plan transformations

  – plan = constructed from solution path in the plan space

➜ Partial-Order Planning is a plan-space planning algorithm that uses partial plans, which are incomplete with respect to the actions contained and with respect to the order of actions.

A least commitment strategy in form of lifting can be used.

Remarks:

❏ Analogously to Lifted Backward Search in state spaces, Lifted Partial-Order Planning uses variables in partial plans in order to describe sets of partial plans that differ only by some constant values.

Plan-Space Planning

Prerequisites for Partial-Order Planning

Definition 10 (Partial Plan, Extension of Partial Plans)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given.

A *partial plan* $p = (A, R, L)$ consists of:

- a finite set A of action instances,
- a finite set R of order constraints $a \prec a'$ defining a strict partial order on the action instances in A, and
- a finite set L of causal links of the form $(a_{\textit{effects}}, l, a_{\textit{precond}})$ such that
  - l is a literal in the language of $\mathcal{S}$,
  - $a_{\textit{effects}} \in A$ is an instance of an action with l as an effect,
  - $a_{\textit{precond}} \in A$ is an instance of an action with l as a precondition, and
  - $a_{\textit{effects}} \prec a_{\textit{precond}}$ is in R.

A partial plan (A', R', L') is an *extension of a partial plan* (A, R, L) if $A \subseteq A'$, $R \subseteq R'$, and $L \subseteq L'$.

Remarks:

- ❑ Plans can contain multiple instances of the same action. Therefore, we have to distinguish these instances in set A.

- ❑ Having $a \prec a'$ as an order constraint means that action a has to be executed before action a'. Therefore, R is a set of constraints for a strict order of the actions in A. R is consistent if there is a strict total order of the actions in A that satisfies the constraints in R. Of course, R is inconsistent if and only if the transitive closure of R contains $a \prec a$ for some $a \in A$.

- ❑ As causal links $(a_{effects}, l, a_{precond})$ justify an execution of $a_{precond}$ after executing $a_{effects}$, causal links should be consistent with the partial order, i.e. $a_{effects} \prec a_{precond}$ should be contained in R for $(a_{effects}, l, a_{precond}) \in L$.

- ❑ In case of Lifted Partial-Order Planning, a partial plan additionally contains
  - – a finite set B of binding constraints of the form $x = y$ and $x \neq y$.

Chapter MK:VI

VI. Planning and Configuration

Plan-Space Planning
Prerequisites for Partial-Order Planning

Definition 11 (Plan Space for Partial-Order Planning)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ and a STRIPS planning problem $(\mathcal{S}, s_{init}, c_{goal})$ be given.

We define actions *start* and *finish* that are new for $\mathcal{S}$:

- *start* has an empty set of preconditions and the set s_{init} as positive effects. According to the CWA we assume that for all positive literals l with $l \notin s_{init}$ the fact $\neg l$ is an effect of *start*.
- *finish* has an empty set of effects and the set c_{goal} as precondition.

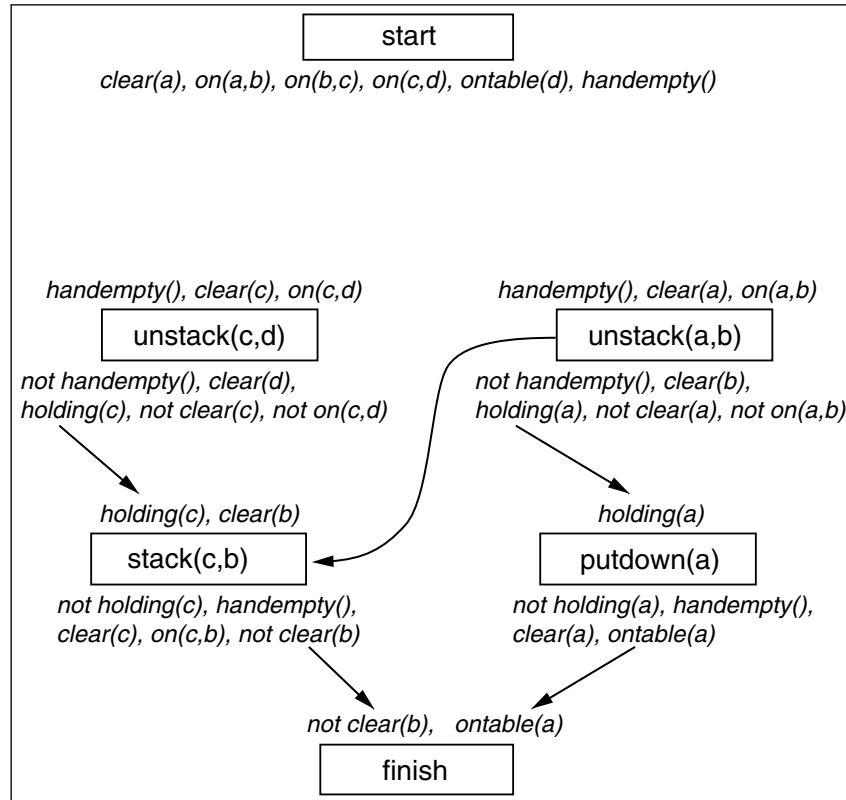The partial plan $p_{init} = (A_{init}, R_{init}, L_{init})$ is defined by

- $A_{init} = \{start, finish\}$,
- $R_{init} = \{start \prec finish\}$, and
- $L_{init} = \emptyset$

The *plan space for partial-order planning* is made up of all partial plans that are extensions p_{init} using only actions that are instances of operators in O.
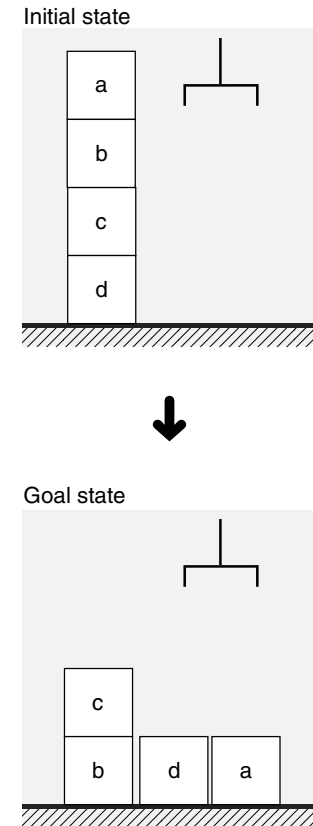
Plan-Space Planning

Example: Representation of a Partial Plan for a Blocksworld Problem

Partial Plan:

start

clear(a), on(a,b), on(b,c), on(c,d), ontable(d), handempty()

handempty(), clear(c), on(c,d)

unstack(c,d)

not handempty(), clear(d), holding(c), not clear(c), not on(c,d)

handempty(), clear(a), on(a,b)

unstack(a,b)

not handempty(), clear(b), holding(a), not clear(a), not on(a,b)

holding(c), clear(b)

stack(c,b)

not holding(c), handempty(), clear(c), on(c,b), not clear(b)

holding(a)

putdown(a)

not holding(a), handempty(), clear(a), ontable(a)

not clear(b),   ontable(a)

finish

Problem:

Initial state

Goal state

Plan-Space Planning

Example: Representation of a Partial Plan for a Blocksworld Problem (continued)

- ❑ Action instantiations A:

    Each action instantiation (including *start* and *finish*) is depicted in a separate box showing operator name and parameter substitution, e.g., $\boxed{unstack(a,b)}$.

    Action preconditions are listed above, action effects below the action box. The goal description is given atop of the *finish* box, the initial state below the *start* box.

    (Negated literals that are true in the initial state because of the CWA are usually not shown, but can be added as needed. For simplicity, we use *not* instead of ¬ for negated literals.)

- ❑ Causal links L:

    Each causal $(a_{effects}, l, a_{precond})$ in L is depicted as an arrow connecting the literal l in the effects listing at the action box corresponding to $a_{effects}$ and the same literal l in the precondition listing at the action box corresponding to $a_{precond}$.

    The causal link expresses that a precondition $a_{precond}$ is satisfied because it is provided by $a_{effects}$.

Plan-Space Planning

Example: Representation of a Partial Plan for a Blocksworld Problem   (continued)

- ❑ Order constraints R:

  - – *start* $\prec$ *finish*, *start* $\prec a$, and $a \prec$ *finish*

    These order constraints are not represented explicitly. By drawing the action box for *start* at the top, the action box for *finish* at the bottom and the boxes for any other action instance a in between the two, these order constraints are expressed.

  - – $a_{effects} \prec a_{precond}$ for $(a_{effects}, l, a_{precond})$ in L

    The order constraint is expressed already by the direction of the arrow depicting the causal link. It is an arrow from a literal l in the effects listing of the action box for $a_{effects}$ to the same literal l in the precondition of the action box for $a_{precond}$.

  - – other order constraints $a \prec a'$

    Such an order constraint is depicted by an arrow directly from the action box for a to the action box of a'.

    These order constraints usually result from removing a threat. Therefore, these arrow are often painted in red and sometimes have a dotted line connecting the arrow with the causal link that was threatened.

Plan-Space Planning

Example: Representation of a Partial Plan for a Blocksworld Problem (continued)

For the partial plan depicted we have

- $A = \{$ *start*, *unstack*(c, d), *unstack*(a, b), *stack*(c, b), *putdown*(a), *finish* $\}$

  (In case of multiple instantiations of the same action, we add indices to the action names.)

- $L = \{$ (*unstack*(c, d), *holding*(c), *stack*(c, b)), (*unstack*(a, b), *holding*(a), *putdown*(a)),
       (*stack*(c, b), ¬*clear*(b), *finish*), (*putdown*(a), *ontable*(a), *finish*) $\}$

- $R = \{$ *start* $\prec$ *finish*,

    start $\prec$ *unstack*(c, d), *unstack*(c, d) $\prec$ *finish*,
    start $\prec$ *unstack*(a, b), *unstack*(a, b) $\prec$ *finish*,
    start $\prec$ *stack*(c, b), *stack*(c, b) $\prec$ *finish*,
    start $\prec$ *putdown*(a), *putdown*(a) $\prec$ *finish*,

    unstack(c, d) $\prec$ *stack*(c, b), *unstack*(a, b) $\prec$ *putdown*(a),
    stack(c, b) $\prec$ *finish*, *putdown*(a) $\prec$ *finish*

    unstack(a, b) $\prec$ *stack*(c, b) $\}$

Plan-Space Planning

Prerequisites for Partial-Order Planning

Definition 12 (Solution Plan for Partial-Order Planning)

A partial plan $p = (A, R, L)$ in the plan space for a STRIPS planning problem $(\mathcal{S}, s_{init}, c_{goal})$ is a *solution plan* for $(\mathcal{S}, s_{init}, c_{goal})$, if and only if

- ❑ the set of order constraints R is consistent, i.e. the transitive closure of R does not contain $a \prec a$ for any $a \in A$, and

- ❑ any total ordering of actions in A conforming to R is a solution plan for $(\mathcal{S}, s_{init}, c_{goal})$.

The transitive closure $\bar{R}$ of a relation R consists of R together with all inequalities $a \prec a'$ that can be derived form inequalities in R by a finite number of applications of the transitivity rule:

If $a \prec a'$ and $a' \prec a''$ holds for some actions a, a', a'', then also $a \prec a''$ holds.

Example: A set R containing $a \prec a'$, $a' \prec a''$, and $a'' \prec a$ is inconsistent.

Plan-Space Planning

Prerequisites for Partial-Order Planning

Definition 13 (Threats and Flaws in Partial Plans)

Let $p = (A, R, L)$ be a partial plan.

An action a_t, $a_t \in A$, is a *threat* on a causal link (a, l, a'), $(a, l, a') \in L$, if

- ❑ a_t has an effect that is inconsistent with l, i.e., $\neg l \in$ *effects*(a_t)
- ❑ and $a \prec a_t$ and $a_t \prec a'$ is consistent with R.

A *flaw* in a partial plan p is

- ❑ either a missing causal link (a, l, a') for a precondition l of some action instance a' in A, i.e., no causal link (a, l, a') with some $a \in A$ is contained in L,
- ❑ or the existence of a threat on a causal link (a, l, a') in L.

For a threat, we have to check whether $R \cup \{a \prec a_t, a_t \prec a'\}$ is consistent or not.

Obviously, a threat can be eliminated by

- ❑ promotion: adding $a' \prec a_t$ to R (applicable only if a' is not *finish*) and
- ❑ demotion: adding $a_t \prec a$ to R (applicable only if a is not *start*).

Plan-Space Planning
Partial-Order Planning

Starting from the initial partial plan, missing causal links are added one after the other and threats resulting from these additions are immediately eliminated.

Nondeterministic algorithm:

1. Start with initial partial plan p_{init} for a STRIPS planning problem.
2. While there are flaws in the current partial plan $p = (A, R, L)$ do:
   (a) Select a precondition l of an action instance a' in A for which there is no causal link in L.
   (b) If no action (including the *start* action) can achieve this subgoal, then return *Failure*.
   (c) Choose an action instance a in A or a new action instance a that achieves this subgoal (i.e. an operator with a ground instantiation that forms an action achieving the subgoal).
   (d) If a is new, add a to A. Add the ordering constraints $a \prec a'$, *start* $\prec a, a \prec$ *finish* to R. Add the causal link (a, l, a') to L.
   (e) Any threat resulting from adding a new action instance or from adding the causal link is eliminated by adding additional ordering constraints to R (choice of promotion or demotion for each threat).
   (f) If R is inconsistent, then return *Failure*.
3. Choose a total ordering of the actions in A conforming to R as a plan p'.
4. Return the plan p'.

Remarks:

❑ Nondeterminism of this algorithm is in the choice of the action instances for missing causal links and in the choice of promotion or demotion for the elimination of threats, but not in the selection of the next missing link to be established. As all missing causal links have to be considered, this is a strategic decision.

❑ The basic step in partial-order planning is
  – establishing a new causal link and
  – eliminating all emerging threats.

  Such a step will eliminate a flaw, but new flaws may arise.

❑ At any point in time before termination (i.e., when the while loop is entered), the partial plan created so far by Partial-Order Planning is consistent and safe.

❑ Implementations of Partial-Order Planning often make use of an agenda that collects all missing causal links in form of pairs (l, a') where a' is an action instance in A and l is a literal in the preconditions of a' for which no causal link exists.

❑ In case of Lifted Partial-Order Planning, inconsistencies in the binding constraints B also lead to *Failure*.

Plan-Space Planning
Prerequisites for Partial-Order Planning

Definition 14 (Safe Plans, Consistent Plans)

A partial plan $p = (A, R, L)$ is *safe* if A contains no threats to causal links in L.

A partial plan $p = (A, R, L)$ is *consistent* if R is consistent.

Lemma 15

A consistent partial plan without flaws that was constructed by *Partial-Order Planning* is a solution plan for the corresponding planning problem.

Proof of the lemma is by induction on the number of actions in A. [Ghallab/Nau/Traverso, p. 93]

Flaws in partial plans can be eliminated by

❑ adding causal links, possibly with new actions, for missing causal links,
❑ adding ordering constraints (adding $a_t \prec a$, i.e. demotion, or adding $a' \prec a_t$, i.e. promotion) in case of threats.

Eliminating flaws may lead to inconsistent plans.

Remarks:

❑ In case of Lifted Partial-Order Planning, a literal l might contain variables. Then, a threat on a causal link (a, l, a') may only be provoked by using a specific substitution. This substitution has to be consistent with the binding constraints.

❑ In case of Lifted Partial-Order Planning, for a solution path the set of binding constraints has to be consistent as well. For the paths defined by a partial plan, all ground instances have to be considered.

❑ In case of Lifted Partial-Order Planning, a threat can be eliminated also by separation: adding binding constraints to B such that the threat is eliminated.

Plan-Space Planning

❑ Plans from Partial Plans

A plan can be generated from a consistent partial plan by a topological sorting of the action instances in A conforming to R.

❑ Partial-Order Planning is *sound*.

If a plan is returned, this plan solves the planning problem at hand.

❑ Partial-Order Planning is *complete*.

If a plan p exists that is a solution to the planning problem, then a partial plan can be found by the non-deterministic algorithm such that (an optimized version of) this plan p corresponds to a topological sorting of the action instances in A conforming to R.

Remarks:

❑ Examples of partial-order planning planning systems:

– J. Scott Penberthy, Daniel S. Weld. *UCPOP: A Sound, Complete, Partial-Order Planner for ADL*, in: Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 92), pp. 103-114

Plan-Space Planning

Example: Partial-Order Planning for the Sussman Anomaly

- ❑ Constants: a, b, c, *floor*
- ❑ Predicates: $on(x, y)$, *clear*(x)
- ❑ Operators:

  operator :   *move*(x, y, z)

      precond : $on(x, y)$, *clear*(x), *clear*(z)

      effects : $\neg on(x, y), \neg$*clear*(z), $on(x, z)$, *clear*(y), *clear*$(floor)$

- ❑ Initial state: $\{on(c, a),$ *clear*$(floor),$ *clear*$(c), on(a,$ *floor*$),$ *clear*$(b), on(b,$ *floor*$)\}$
- ❑ Goal: $\{on(a, b), on(b, c)\}$

- ❑ Initial partial plan:

  - – Additional actions:

    action :   *start*$()$

        precond :

        effects : $on(c, a),$ *clear*$(floor),$ *clear*$(c), on(a,$ *floor*$),$ *clear*$(b), on(b,$ *floor*$)$

    action :   *finish*$()$

        precond : $on(a, b), on(b, c)$

        effects :

  - – $A = \{$*start*$(),$ *finish*$()\}$
  - – $R = \{$*start*$() \prec$ *finish*$()\}$
  - – $L = \emptyset$

Example: Partial-Order Planning for the Sussman Anomaly (continued)
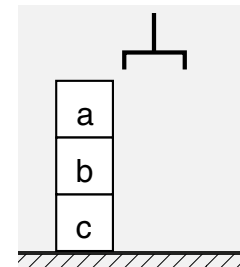
start

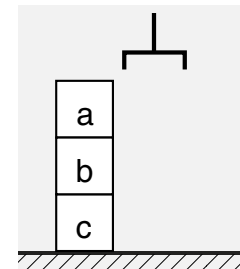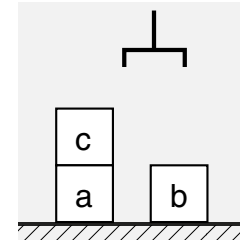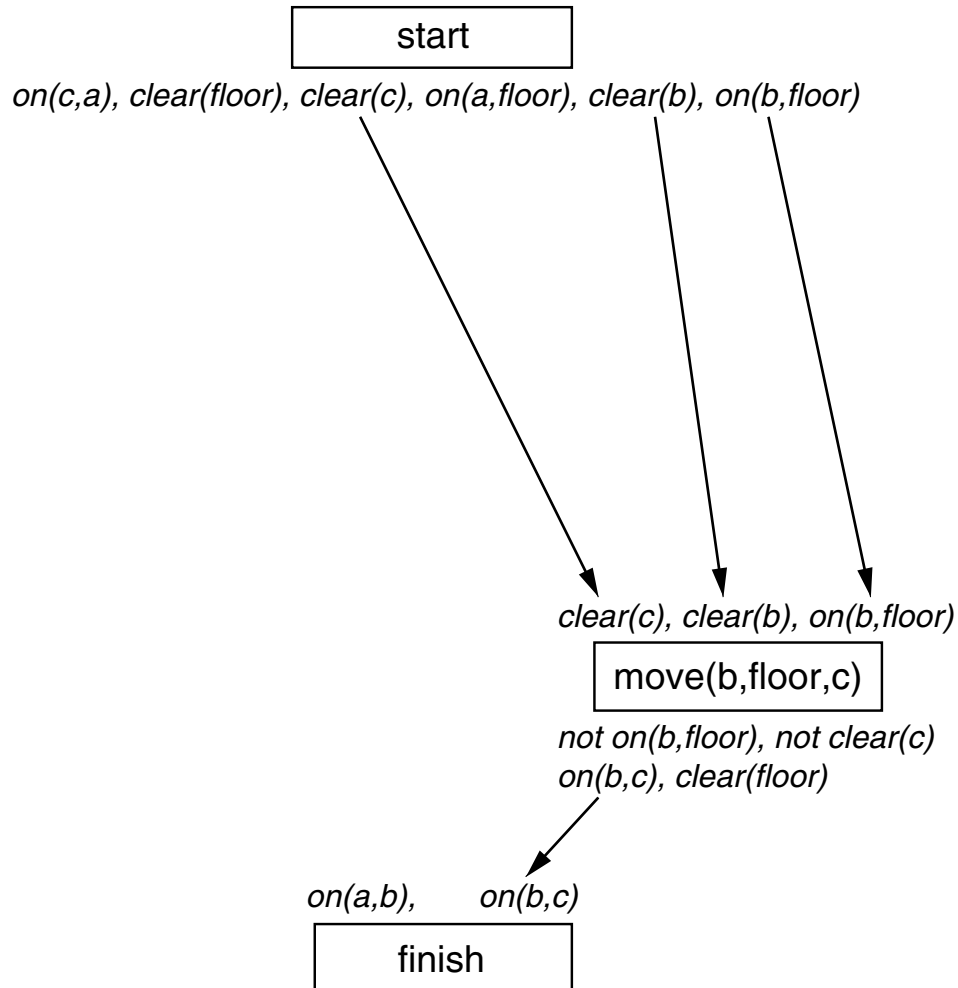*on(c,a), clear(floor), clear(c), on(a,floor), clear(b), on(b,floor)*
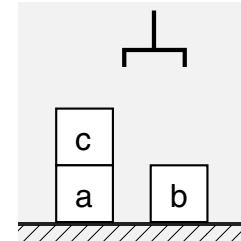
on(a,b),   on(b,c)

finish

Plan-Space Planning

start

on(c,a), clear(floor), clear(c), on(a,floor), clear(b), on(b,floor)

clear(c), clear(b), on(b,floor)

move(b,floor,c)

not on(b,floor), not clear(c)
on(b,c), clear(floor)

on(a,b),   on(b,c)

finish

Plan-Space Planning

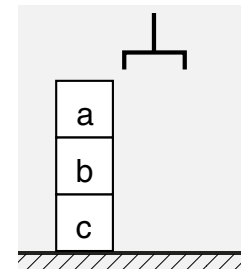Example: Partial-Order Planning for the Sussman Anomaly   (continued)

start

on(c,a), clear(floor), clear(c), on(a,floor), clear(b), on(b,floor)

move(a,floor,b)
threatens clear(b)

⇒ Resolved by
  putting behind
  move(b,floor,c)

clear(a), on(a,floor), clear(b)   clear(c), clear(b), on(b,floor)

move(a,floor,b) ⟵ move(b,floor,c)

not on(a,floor), not clear(b)   not on(b,floor), not clear(c)
clear(floor), on(a,b)   on(b,c), clear(floor)

on(a,b),   on(b,c)

finish

Plan-Space Planning

Example: Partial-Order Planning for the Sussman Anomaly   (continued)

start

on(c,a), clear(floor), clear(c), on(a,floor), clear(b), on(b,floor)

on(c,a), clear(c), clear(floor)

move(c,a,floor)

not on(c,a), not clear(floor)
clear(a), on(c,floor), clear(floor)

clear(a), on(a,floor), clear(b)

move(a,floor,b)

not on(a,floor), not clear(b)
clear(floor), on(a,b)

clear(c), clear(b), on(b,floor)

move(b,floor,c)

not on(b,floor), not clear(c)
on(b,c), clear(floor)

on(a,b),   on(b,c)

finish
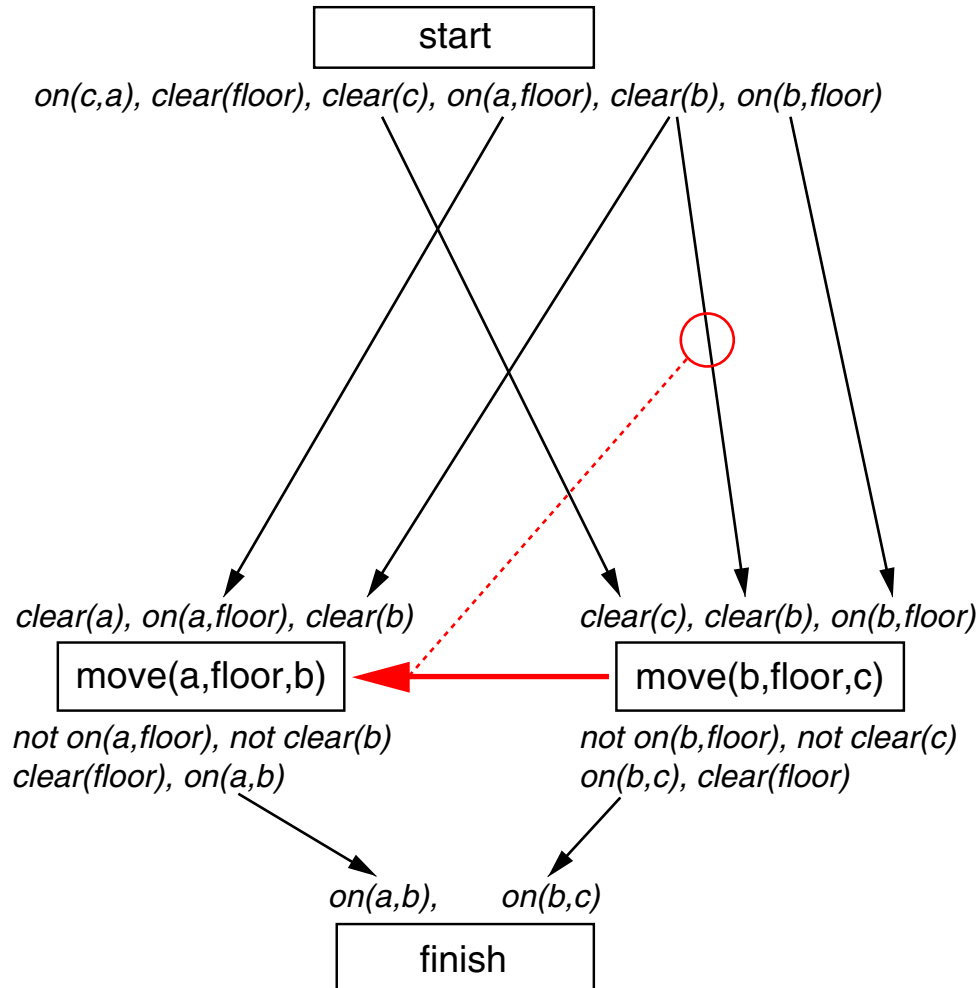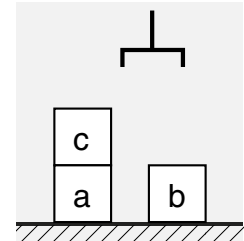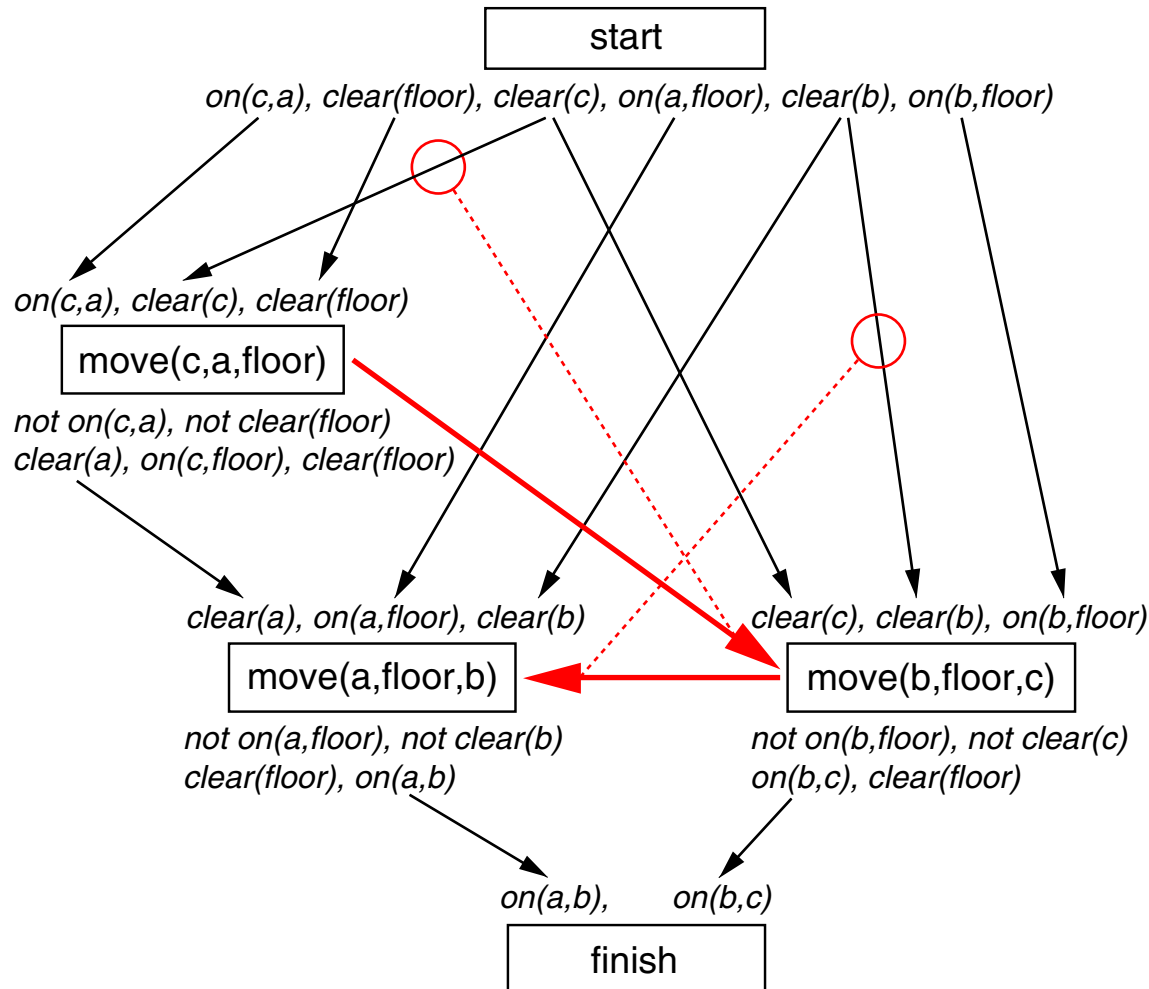
move(a,floor,b)
threatens clear(b)

⇒ Resolved by
   putting behind
   move(b,floor,c)

move(b,floor,c)
threatens clear(c)

⇒ Resolved by
   putting behind
   move(c,a,floor)

Plan-Space Planning

Solution path in the plan space:

Start Node

Goal Node

Chapter MK:VI

HTN Planning

Prerequisites

❑ Objective of HTN Planning is to accomplish given tasks.

Definition 16 (Task)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given. A *task* $t(p_1, \ldots, p_n)$ is specified by

❑ a task name t that is either an operator name in O or new for $\mathcal{S}$ and

❑ a parameter list $p_1, \ldots, p_n$ consisting of constants in C or variables (where n is the corresponding arity for operators in O).

$t(p_1, \ldots, p_n)$ is a *primitive task* if t an operator name in O. Otherwise, $t(p_1, \ldots, p_n)$ is a *complex task*.

A task is *ground* if its parameter list does not contain a variable.

An action a *accomplishes a task* t in a state s if $a = t$ and a is applicable in s.

➡ Only primitive ground tasks can be accomplished by actions.

HTN Planning

❏ In HTN Planning, accomplishing tasks is constrained.

Definition 17 (Task Network)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given. A *task network* N is a pair $N = (T_N, C_N)$ where

❏ T_N is a finite set of tasks and
❏ C_N is a finite set of constraints of the form

$$
\begin{array}{ll}
t \prec t' & \text{with } t, t' \in T_N, \\
before(T'_N, l) & \text{with } T'_N \subseteq T_N \text{ and a literal } l \text{ from } \mathcal{S}, \\
after(T'_N, l) & \text{with } T'_N \subseteq T_N \text{ and a literal } l \text{ from } \mathcal{S}, \text{ or} \\
between(T'_N, T''_N, l) & \text{with } T'_N, T''_N \subseteq T_N \text{ and a literal } l \text{ from } \mathcal{S}.
\end{array}
$$

A *simple task network* has only precedence constraints.

A task network is *ground* if all its tasks are ground. A task network is *primitive* if all its tasks are primitive.

➜ Only primitive ground task networks can be accomplished by plans.

Remarks:

❑ The intended meaning of the constraints is as follows:

- $t \prec t'$:

Accomplishing t has to be finished before starting to accomplish t'.

- *before*(T', l):

In the state immediately before starting to accomplish T', l has to be valid.

- *after*(T', l):

In the state immediately after finishing to accomplish T', l has to be valid.

- *between*(T', T'', l):

In the state immediately after finishing to accomplish T' and just before starting to accomplish T'', l has to be valid.

HTN Planning

❑ In HTN Planning, complex tasks can be decomposed.

Definition 18 (Method, HTN Model, HTN Planning Problem)

Let a STRIPS model $\mathcal{S} = (C, P, O)$ be given.

A *method* $m(x_1, \ldots, x_n)$ is a triple (t_m, T_m, C_m) where

❑ t is a task,
❑ (T_m, C_m) is a task network, and
❑ $x_1, \ldots, x_n$ is the list of variables occurring in t_m or (T_m, C_m).

A *HTN model* $\mathcal{H} = (C, P, O, M)$ is given by a STRIPS model $\mathcal{S} = (C, P, O)$ and a finite set M of methods.

A *HTN planning problem* $(\mathcal{H}, s_{init}, N_{goal})$ consists of an HTN model $\mathcal{H}$, an initial state s_{init} and a task network N_{goal} with task names from $\mathcal{H}$.

➜ Methods can be used to decompose tasks in N_{goal} into a network of primitive tasks.

HTN Planning

Prerequisites (continued)

Definition 19 (HTN Space)

Let a *HTN model* $\mathcal{H} = (C, P, O, M)$ and a *HTN planning problem* $(\mathcal{H}, s_{init}, N_{goal})$ be given.

A method $m = (t_m, T_m, C_m)$ is applicable to a task t in a task network $N = (T_N, C_N)$ if there is a substitution σ such that $\sigma(t_m) = t$. The *successor task network* $N' = (T'_N, C'_N)$ is defined by $T'_N = (T_N \setminus \{t\}) \cup \sigma(T_m)$ and C'_N computed from C_N by

- ❑ replacing order constraints $t' \prec t$ and $t \prec t'$ by all corresponding order constraints with t substituted by tasks from $\sigma(T_m)$,
- ❑ replacing state constraints *before*(T', l), *after*(T', l), and *between*(T', T'', l) by corresponding state constraints with T' substituted by $T' \setminus \{t\}) \cup \sigma(T_m)$ resp. T'' substituted by $T'' \setminus \{t\}) \cup \sigma(T_m)$.

The *HTN space* for $(\mathcal{H}, s_{init}, N_{goal})$ is the set of all task networks that can be constructed by applying methods in M.

➜   In the HTN space we have to look for task networks that define solution plans.

HTN Planning

❑ In HTN Planning, a plan has to accomplish tasks taking into account the constraints.

Definition 20

Let a HTN planning problem $(\mathcal{H}, s_{init}, N_{goal})$ be given and let $N_{goal} = (T_N, C_N)$ be a primitive ground task network. Let $(t_1, \ldots, t_k)$ be an ordering of the tasks in T_N that is consistent with C_N. So, $p = (t_1, \ldots, t_k)$ is a plan.

Plan p *accomplishes the tasks in task network* N_{goal} if p is applicable in state s_{init} and the constraints in C_N are met, i.e. for $T'_N, T''_N \subseteq T_N$ and a literal l we have

before(T', l)	l is valid directly before executing the first action in T'_N,
after(T', l)	l is valid directly after executing the last action in T'_N,
between(T', T'', l)	l is valid directly before executing the first action in T'_N and l is valid directly after executing the last action in T'_N.

➜ For primitive ground task networks a total ordering and a constraint check is needed.

HTN Planning

❑ In HTN Planning, non-primitive task networks have to be decomposed.

Definition 21 (Solution Plan)

Let a *HTN model* $\mathcal{H} = (C, P, O, M)$ and a HTN planning problem $(\mathcal{H}, s_{init}, N_{goal})$ be given and let $N_{goal} = (T_N, C_N)$ be the task network.

If N_{goal} is primitive and if there is a ground substitution σ and an ordering $p = (\sigma(t_1), \ldots, \sigma(t_k))$ of the tasks in $\sigma(T_N)$ such that p is consistent with $\sigma(C_N)$ and p accomplishes the tasks in task network $\sigma(N_{goal})$, then p is a *solution plan* for $(\mathcal{H}, s_{init}, N_{goal})$.

If N_{goal} is non-primitive and N_{goal} can be decomposed by a finite number of method applications from M into a primitive task network (T'_N, C'_N) for which a solution plan p exists, then p is a solution plan for $(\mathcal{H}, s_{init}, N_{goal})$.

➜ In HTN Planning we have to search the HTN space for task networks that have solution plans.

HTN Planning
Abstract HTN Planning

Starting from the task network N_{goal} from the HTN planning problem, decompose non-primitive tasks, instantiate and order the primitive tasks forming a plan, and check the plan.

Nondeterministic algorithm:

1. Start with task network N_{goal} from a HTN planning problem.
2. While the current task network $N = (T_N, C_N)$ is not primitive do:
   (a) Select a non-primitive task t.
   (b) If no ground instance of a method can decompose this task, then return *Failure*.
   (c) Choose a ground method instance m_g.
   (d) Decompose t using method m_g.
3. If there in no ordering of the actions in the primitive task network N that is consistent with C_N, then return *Failure*.
4. Choose such a total ordering as a plan p.
5. If p is inconsistent with the state constraints in C_N, then return *Failure*.
6. Return the plan p.

Problem: Which decompositions are promising?

Remarks:

- ❑ Task decomposition in HTN Planning can be cyclic: A task that was decomposed can reappear in the decomposition.

HTN Planning
Simplification of HTN Planning

❑ In HTN Planning, a plan has to accomplishing tasks taking into account the constraints.

Definition 22 (Total-order STN Planning Problem)

Let a HTN planning problem $(\mathcal{H}, s_{init}, N_{goal})$ be given and let $N_{goal} = (T_N, C_N)$ be a primitive ground task network.

Further let the task network N_{goal} and the task networks (T_m, C_m) of the methods $m = (t_m, T_m, C_m)$ in M bet totally ordered. Apart from ordering constraints, let C_m contain only constraints *before*(T_m, l) with literals l.

Then, $(\mathcal{H}, s_{init}, N_{goal})$ is called a total-order STN planning problem.

➜ Total-order task networks can be specified as lists of tasks. So, a method in a total-order STN planning problem can be described as

$$m = (t_m, \textbf{\textit{precond}}(m), (t_{(m,1)}, \ldots, t_{(m,k_m)}))$$

Remarks:

❑ STN stands for Simple Task Networks.

HTN Planning
Total-Order Forward Decomposition HTN Planning

Starting from the task network N_{goal} from the total-order STN planning problem, the first task if decomposed in case of a non-primitive task or applied to the current state in case of a primitive task.

Nondeterministic algorithm:

1. Start with the list of tasks in network N_{goal} from a total-order STN planning problem and an empty plan p and the state s_{init}.

2. While the current list of tasks is not empty do:

   (a) If t_1 is a non-primitive task and if no ground instance of a method can decompose this task, then return *Failure*.

   (b) Otherwise, if t_1 is a non-primitive task, choose a ground method instance m_g and decompose t_1 using method m_g.

   (c) Otherwise, if t_1 is a primitive task and if t_1 is not applicable to the current state, then return *Failure*.

   (d) Otherwise, if t_1 is a primitive task, append t_1 to the plan p and let the successor state be the current state.

3. Return the plan p.

Remarks:

❏ For Abstract HTN Planning and Total-Order Forward Decomposition HTN Planning, we can define also versions using lifting.

❏ In Total-Order Forward Decomposition HTN Planning a plan is constructed. Similarly, a partial planning approach is possible when dealing with simple task networks for which only a partial order of tasks is given.

❏ Examples of HTN planning planning systems:

  – D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. *SHOP: Simple Hierarchical Ordered Planner*, in: Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI 99), pp. 968-973
  – D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. *SHOP2: An HTN Planning System*, in: J. Artif. Intell. Research **20**, (2003), pp. 379-404

❏ Further reading:

  – K. Erol, J. Hendler, D. S. Nau. *Complexity Results for HTN Planning*, in: Annals of Mathematics and Artificial Intelligence **18** (1996), pp. 69-93

HTN Planning
Total-Order Forward Decomposition vs. Abstract HTN Planning

- ❏ Total-Order Forward Decomposition can detect early that a plan will not be applicable to the initial state.

- ❏ Both algorithms are *sound*.
  By definition of solution plans, the plans returned solve the planning problem at hand.

- ❏ Both algorithms are *complete*.
  A sequence of task decompositions for a total-order STN planning problem can be rearranged in such a way that always a first non-primitive task is decomposed. The resulting task network is a plan that is not changed by that rearrangement. Applicability of the plan is checked by the algorithm.

- ❏ There are undecidable problems that can be expressed as HTN planning problems, even as STN planning problems (i.e. allowing task networks that are not totally ordered). Therefore, the Plan Existence Problem for STN Planning and HTN Planning is strictly semi-decidable.

Remarks:

❑ Since undecidable problems can be encoded as HTN/STN planning problems, even sound and complete HTN/STN planning algorithms will not terminate for all inputs.