Automated Planning

- Automated Planning - Characteristics
- Linear Planning in STRIPS
- Non-Linear Planning in TWEAK

Introduction

- An important aspect of common sense reasoning and of automated planning problems is the capacity to represent actions and to reason about these actions, so to reason about changes
- Automated planning, which implies using actions, represents an important domain of artificial intelligence

Introduction

- Most planning problems refer to complex domains, for which the representation and the processing of complete description of the problem states can become difficult
- This is the reason why, in many cases, it is necessary to use a strategy to represent partially the problem universe and also a problem decomposition into subproblems strategy

Characteristics

- Reasoning about actions implies solving three common sense reasoning problems:
  - Frame problem
  - Qualification problem
  - Branching problem
- Consider the case of a robot which must execute the actions to move objects from a room

Frame Problem

- If the robot moves the table from the center of the room near to the window, an automated planning system must be capable to deduce the fact that neither the carpet, nor the wardrobe do not change their position, even if this is not explicitly defined
- This is the **frame problem**, that is the identification and the deduction of all the facts which are not changed as an effect of executing an action or of time passing

Qualification Problem

- If the robot must execute the action of moving the wardrobe from the room, it is possible to not succeed, either because the wardrobe is too heavy or because it is fixed on the floor
- Recording all the necessary preconditions to execute an action is impossible in most cases – this is the **qualification problem**

Branching Problem

- If the robot moves the wardrobe, the objects on the wardrobe are moved in the same time, some parts of the carpet will be blocked, it is possible that the window will be blocked
- This is the **branching problem**, because it is impossible to record all the consequences of an action

Observations

- The complete and explicit representation of all the states for an action execution problem in the real world is almost impossible for complex situations
- Because of the complexity of planification problems, in many cases it is better to use the problem decomposition into subproblems

Problem Decomposition

- The problem solving techniques for decomposing a problem into subproblems are useful only for problems in which the subproblems are not interacting between them, that is they are **independent**
- Solving a subproblem does not depend on solving other subproblems and it is not affected by solving the next subproblems

Observation

- For many problems there is no decomposition into subproblems which are independent
- The subproblems obtained by decomposition are **interdependent**

Observations

- From the set of interdependent problems, it is possible to identify the **almost independent** problems, derived from a problem decomposition, between which there is just a weak interaction
- This characteristic simplifies the problem of automated plan generation, but it is still necessary to consider the possible interactions

Observations

- Planning implies the establishment of a sequence of actions to reach a certain goal
- In the case of solving a planification problem, the distinction between plan and action is not so clear, because the plan proposed to reach a certain goal is not applied, in most cases, while the plan is developed

Observations

- Even if the problem implies irrevocable actions in the real world, the planification program can simulate the plan, which allows to use a tentative strategy
- The success of this approach is based on the assumption that the real world is predictable
- Unfortunately, this is not always true, so the simulation process can not consider all the possible alternatives

Observations

- An important aspect of planification problems is that the real world is not always predictable
- A good plan for reaching a goal in certain conditions can become a bed plan when some real conditions are different
- The unexpected change of the contest can invalidate all the plan or just a part of it

Observations

- An important problem is to reconstruct the plan, keeping those plan parts which still remain valid
- This aspect is an argument for problem decomposition into subproblems

Observations

- Problem decomposition into subproblems reduces the complexity of dynamic plan revision operations, which are necessary during the plan execution in an unpredictable world
- Plan revision operations are executed easier in case of independent or almost independent problems

Characteristics

- Planning and reasoning about actions problems need a search process
- In most cases, this is a goal driven search, which starts from the goal state and determines the sequences of operators which transforms the initial state into the final state

Observations

- To facilitate the plan revision operations, it is useful to associate to each action executed in a plan, the reason which determines the action in the plan
- If in a new situation, a certain action is no more necessary or valid, it is possible to determine the remaining actions from the plan, which depends on the given action, so it is possible to identify the actions which must be reconsidered

Planification Methods

- Automated plan generation methods can be divided into two categories:
  - Linear planning methods
  - Non-linear planning methods
- In the case of linear planning, a sequence of goals is solved by satisfying each subgoal
- A plan generated by such a method contains a sequence of actions which satisfies the first subgoal, then the second subgoal aso

Planification Methods

- This method can not be used for solving planification problems in which the subproblems interact
- In this case, a non-linear planning method is used, which generates plans by considering simultaneously many subproblems, obtained from the initial problem decomposition
- In the beginning, the plan is incompletely specified and is improved subsequently, by considering the possible interactions

Observations

- Such a method is called non-linear planning because the plan is not composed from a linear sequence of complete subplans
- The non-linear planning implements a problem solving paradigm known as the **delayed decisions strategy**
- This approach is also useful in the case of dynamic plan revision

Example

- Consider the robot Robo, which uses the means and analysis to develop a planning trip between Paris and Bucharest
- There are many possibilities to travel between the two cities and Robo must select a possibility

Example

- The operators used by Robo are:
  - use_plane(x,y)
  - use_train(x,y)
  - use_car(x,y)
  - use_taxi(x,y)
  - walk(x,y)

  With the preconditions and the effects specified

Example

Operator	**Preconditions**	**Effect**
use_plane(x,y)	is_at(Robo,x)	is_at(Robo,y)
use_train(x,y)	is_at(Robo,x)	is_at(Robo,y)
use_car(x,y)	is_at(Robo,x)	is_at(Robo,y)
use_taxi(x,y)	is_at(Robo,x)	is_at(Robo,y)

Example

- The difference table for the planning trip problem

Operatori / Diferenþe	foloseºte_ avion	foloseºte_ tren	foloseºte_ masinã	foloseºte_ taxi	mers
Mai mult de 2000 km	*				
Între 100-2000 km		*	*		
Între 1-100 km				*	
Sub 1 km					*
Precondiþii	Robo la avion	Robo la tren	Robo la maºinã	Robo la taxi	

Example

- To go from Paris to Bucharest, the initial distance is grater than 2000 km, so the selected operator is use_plane
- To use the plane, Robo must be near the plane
- Supposing that Robo in the in the center of Paris and the distance to the airport is between 1-100 km, Robo at the plane is a subgoal which can be fulfilled if the operator use_taxi is applied

Example

- The use the taxi, Robo must be near to the taxi and Robo can walk to reach the taxi
- This subgoal has no initial condition, so this subgoal can be directly satisfied
- The same reasonning can be applied when Robo arrives at the airport in Bucharest and than to arrive in the center of the city

Observations

- This approach is not good to solve complex problems, because on one hand, the elimination of a difference can influence the established plan to eliminate other difference and, on the other hand, the difference table can have huge dimensions for complex problems

Linear Planning in STRIPS

- Actions representation and plan building in STRIPS start from the high hypothesis that the problem domain is not significantly changed by executing an action
- This is the reason why, a unique model of the problem universe is used in STRIPS, which is permanently updated to express the result of the executed actions
- The current state of the problem universe is described by a set of well formed formulas in first order predicate logic

Linear Planning in STRIPS

- Also, there are some domain specific axioms
- An action description is done by specifying the conditions in which this action can be performed and the changes which appear in the problem as an effect of executing the action
- The changes generated by performing an action can be seen as being formed by two sets:
  - A set of formulas to describe the new problem state, which become true after the action is executed
  - A set of formulas which become false after the action is executed

Linear Planning in STRIPS

- Each operator is defined by the following elements:
  - **Action** – represents the action associated to the operator
  - **Precondition List (PL)** – contains the formulas which must be true in a problem state, such that the operator can be applied
  - **Addition List (AL)** – contains the formulas which will become true after the operator is applied
  - **Elimination List (EL)** – contains the formulas which will become false after the operator is applied

Linear Planning in STRIPS

- To each action is attached a set of precondition
- STRIPS uses the means end analysis as a strategy for solving problems
- Instead of representing the results of an action by the complete description of the new state, STRIPS represents only the changes produced by the action to the current state

Linear Planning in STRIPS

- This allows, on one hand, the description simplification and, on the other hand, frame problem solving
- Any predicate which does not explicitly appear in **AL** or in **EL** associated to an action, is considered unchanged by executing that action

Example

- Consider the blocks world – there a flat surface, called table on which some cubes can be placed, all having the same dimensions

Example

- The problem asks to generate a transformation plan for the initial configuration into the final configuration, knowing the following two conditions:
  - 1. The cubes can be placed one over another using a robot arm to move the cubes
  - 2. The robot arm can hold only one cube at a given time

Actions

- The robot arm can perform the following actions:
- **UNSTACK(A,B)** – represents the action of taking the block A, placed over the block B. To execute the action, the robot arm must be empty and over the block A there must be no other blocks

Actions

- **STACK(A,B)** – represents the action of placing the block A over the block B. To execute the action, the robot arm must already hold the block A and over the block B there must be no other blocks
- **PICKUP(A)** – represents the action of taking the block A, placed on the table. To execute the action, the robot arm must be empty, the block A must be on the table and over the block A there must be no other blocks

Actions

- **PUTDOWN(A)** – represents the action of placing the block A on the table. To execute the action, the robot arm must hold the block A
- To specify the conditions in which an operator can be executed and the result of executing the action, the following predicates are defined:

Predicates

- **ON(A,B)** is true if the block A is over the block B
- **ONTABLE(A)** is true if the block A is on the table
- **CLEAR(A)** is true if there is no block placed over the block A
- **ARMEMPTY** is true if the robot arm is empty

Axioms

- The domain axioms that is logical true assertions in the blocks world are:

$$(\exists x)\ (\mathrm{HOLD}(x)) \rightarrow \sim \mathrm{ARMEMPTY}$$

$$(\forall x)\ (\mathrm{ONTABLE}(x) \rightarrow \sim (\exists y)\ (\mathrm{ON}(x, y)))$$

$$(\forall x)\ (\sim (\exists y)\ (\mathrm{ON}(y, x)) \rightarrow \mathrm{CLEAR}(x))$$

Operators

- The operators in STRIPS are:

STACK(x,y)	PL:	$CLEAR(y) \wedge HOLD(x)$
	EL:	$CLEAR(y) \wedge HOLD(x)$
	AL:	$ON(x,y) \wedge ARMEMPTY$
UNSTACK (x,y)	PL:	$ON(x,y) \wedge CLEAR(x) \wedge ARMEMPTY$
	EL:	$ON(x,y) \wedge ARMEMPTY$
	AL:	$HOLD(x) \wedge CLEAR(y)$
PICKUP(x)	PL:	$CLEAR(x) \wedge ONTABLE(x) \wedge ARMEMPTY$
	EL:	$ONTABLE(x) \wedge ARMEMPTY$
	AL:	$HOLD(x)$
PUTDOWN (x)	PL:	$HOLD(x)$
	EL:	$HOLD(x)$
	AL:	$ONTABLE(x) \wedge ARMEMPTY$

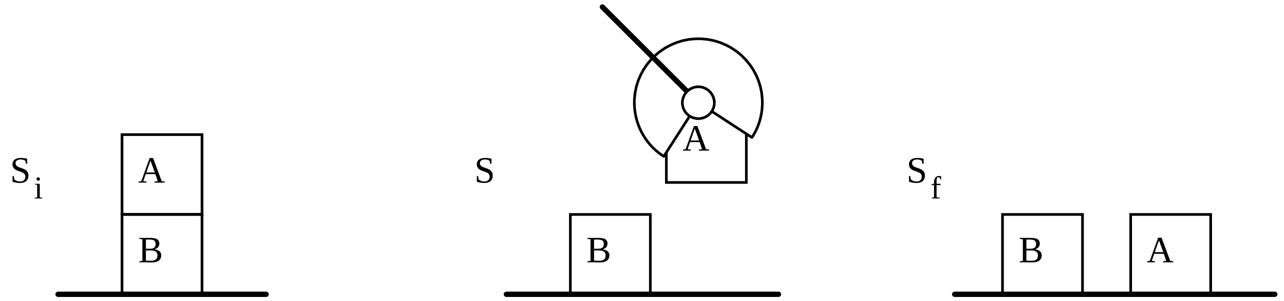
Observations

- For simple actions, PL is identical to EL, but the preconditions are not always eliminated
- For instance, if the robot arm takes a block (PICKUP(x)), the block must not have another block over it. After the robot arm takes the block, this still does not have another block over it

Plan Execution

- For a given problem, the transition from the initial state S_i to the final state S_f, going through the intermediate state S is represented, in which the block B is free on the table and the block A is hold by the robot arm

Plan Execution



$$\text{ONTABLE(B)} \wedge \text{ON(A,B)} \wedge \text{CLEAR(A)} \xrightarrow[\text{UNSTACK(A,B)}]{} $$

$$\text{CLEAR(A)} \wedge \text{ONTABLE(B)} \wedge \text{CLEAR(B)} \wedge \text{HOLD(A)} \xrightarrow[\text{PUTDOWN(A)}]{} $$

$$\text{CLEAR(A)} \wedge \text{CLEAR(B)} \wedge \text{ONTABLE(A)} \wedge \text{ONTABLE(B)} \wedge$$

Plan Synthesis in STRIPS

- The final state description is considered, the final state assertions which do not exist in the initial state are inspected and the operators (actions) for which AL contain the assertions from the final state are selected
- To apply such an operator, the formulas from PL must be true in the current state
- The system verifies the preconditions validity, using the facts from the current state and the domain axioms

Plan Synthesis in STRIPS

- Preconditions which are not fulfilled become, in their turn, assertions which must be satisfied in order to apply the operators
- If all the preconditions are true, the operator can be applied
- Before applying the operator, its variables are instantiated, if it is necessary, using the substitutions performed while the operator preconditions are proved

Plan Synthesis in STRIPS

- When an operator is applied, from the current state are eliminated the formulas which identify with a formula from EL and the formulas from AL are added, such that a new state is obtained
- The process continues until a sequence of operators is found, which can transform the description of the initial state into the description of the final state
- This operators sequence represents the plan synthesis

Observations

- Such a problem solving process implies a backtracking procedure
- Preconditions fulfillment for an operation establishes the goal list which must be satisfied and a certain goal can be fulfilled in different ways
- Because there can be search paths which can fail, it is necessary to use a backtracking procedure

Observations

- Backtracking to a previous state, from a current state obtained based on the changes from AL and EL is performed by adding to the current state the elements from EL and eliminating from the current state the elements from AL
- To make this possible, it is necessary to memorize the applied actions and the objects upon which this action is applied, for each state obtained during search

Example

- To backtrack from the final state S_f in the initial state S_i , first the effects of the action PUTDOWN(A) must be eliminated, then the effects of the action UNSTACK(A,B)
- The backtracking process can be improved by adding a system for keeping the data consistency

Disadvantages

- One problem that can appear is a loop when satisfying goals
- To satisfy the goal G_1 it is necessary to fulfill the precondition G_2 and to satisfy goal G_2 it is necessary to fulfill the precondition G_3 and to satisfy goal G_3 it is necessary to fulfill precondition G_1
- This situation is equivalent to the generation of a graph search space with loops

Disadvantages

- The second problem refers to the interaction between goals
- If it is necessary to satisfy a list of goals and after the goals G_1 and G_2 are fulfilled, when satisfying the goal G_3, the goal G_1 is invalidated and the system can not satisfy the conjunction of goals G_1, G_2,…, G_n
- There is an interaction between the goals G_1 and G_3 on the path G_1, G_2, G_3

Solutions

- The goals are rearranged if it is possible, such that the interaction dissappear
- Another solution is searched, hopping that, for the new plans, the goal G_1 will not be invalidated
- The goal G_1 is satisfied under the conditions that the goals G_2 and G_3 are fulfilled
- There are situations in which none of the above solutions is feasible

Observations

- For a real time program, a goal which was satisfied can not be fulfilled any more or the actions effect can not be cancelled
- A plan which contains an action like “destroy the door” can not be changed if the door does not exist any more

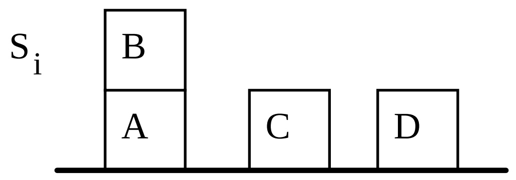
Example

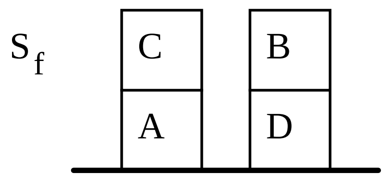
- If in a sequence of goals
  - G_1: Enter in the house
  - G_2: Keep the door integrity
- The goal G_1 can be satisfied aternativelly by fulfilling the subgoal G_{11} or the subgoal G_{12}
  - G_{11}: Open the door
  - G_{12}: Destroy the door
- If the subgoal G_{11} fails and the subgoal G_{12} is executed, then the goal G_2 can not be satisfied anymore

STRIPS System Running

- It is necessary to find an actions plan to arrive from the initial state S_i into the final state S_f
- The abbreviation OTAD is used for the goals conjunction ONTABLE(A) $\wedge$ ONTABLE(D), because these goals of the final state are already fulfilled in the initial state
- Depending on the order of goal fulfillment, from the initial state it is possible to create two stacks of goals

STRIPS System Running





$ON(B,A) \wedge$
$ONTABLE(A) \wedge$
$ONTABLE(C) \wedge$
$ONTABLE(D) \wedge$
$ARMEMPTY$

$ON(C,A) \wedge$
$ON(B,D) \wedge$
$ONTABLE(A) \wedge$
$ONTABLE(D)$

Stack 1	**Stack 2**
$ON(C,A)$	$ON(B,D)$
$ON(B,D)$	$ON(C,A)$
$ON(C,A) \wedge ON(B,D) \wedge OTAD$	$ON(C,A) \wedge ON(B,D) \wedge OTAD$

STRIPS System Running

- Suppose that Stack 1 is investigated
- Stack 2 will find quicker the solution, but it does not show the previous problems

STRIPS System Running

- The goal on ON(C,A) is tried to be satisfied
- The suitable operator is STACK(C,A)
- This operator replaces the goal ON(C,A) in the stack because if this operator is applied, the result will be ON(C,A)

STRIPS System Running

- To apply the operator STACK(C,A) , its preconditions must be fulfilled
- These become goals, in their turn, and are introduced in the stack
- The order of introducing the goals in the stack is important
- It is better that the goal HOLD(C) will be the last one, using the heuristic which says “in order to execute other actions, the robot must have the arm empty”

STRIPS System Running

/* to fulfiil goal ON(C,A) */

STACK(C, A)
ON(B,D)
$ON(C,A) \wedge ON(B,D) \wedge OTAD$

/* preconditions of the operator STACK(C,A)*/

CLEAR(A)

HOLD(C)

$CLEAR(A) \wedge HOLD(C)$
STACK(C, A)

ON(B,D)

$ON(C,A) \wedge ON(B,D) \wedge OTAD$

STRIPS System Running

- The goal CLEAR(A) is verified
- This goal is not satisfied in the state S_i due to the axiom:

$$(\forall x)\ (\sim (\exists y)\ (ON(y,x)) \rightarrow CLEAR(x))$$

STRIPS System Running

- The goal is fulfilled by applying the operator UNSTACK(B,A)
- This will replace the goal CLEAR(A) in the stack

STRIPS System Running

$\mathrm{ON(B,A)}$

$\mathrm{CLEAR(B)}$

$\mathrm{ARMEMPTY}$

$\mathrm{ON(B,A) \wedge CLEAR(B) \wedge ARMEMPTY}$

$\mathbf{UNSTACK(B,A)}$

$\mathrm{HOLD(C)}$

$\mathrm{CLEAR(A) \wedge HOLD(C)}$

$\mathbf{STACK(C,A)}$

$\mathrm{ON(B,D)}$

$\mathrm{ON(C,A) \wedge ON(B,D) \wedge OTAD}$

STRIPS System Running

- The goal ON(B,A) is satisfied in the initial state and is eliminated from the stack
- The same for the goals CLEAR(B) and ARMEMPTY
- The preconditions for the operation UNSTACK(B,A) are verified to avoid the situation in which a goal satisfaction will invalidate another goal
- The goal conjunction is satisfied

ON(B,A) $\wedge$ CLEAR(B) $\wedge$ ARMEMPTY

STRIPS System Running

- In these conditions, the operator UNSTACK(B,A) can be applied and the current state S_i is modified, obtaining a new state S_1

$$S_i \rightarrow S_1\text{: } ONTABLE(A) \wedge ONTABLE(C) \wedge ONTABLE(D) \wedge HOLD(B) \wedge CLEAR(A)$$

$$Plan = (UNSTACK(B,A))$$

STRIPS System Running

- The stack becomes:

$$\begin{array}{l} HOLD(C) \\ \qquad CLEAR(A) \wedge HOLD(C) \\ \qquad \mathbf{STACK(C,A)} \\ \qquad ON(B,D) \\ \qquad ON(C,A) \wedge ON(B,D) \wedge OTAD \end{array}$$

STRIPS System Running

- To satisfy HOLD(C), two operators can be applied: PICKUP(C) end UNSTACK(C,y), so there are two alternative branches of the search tree
- If the alternative UNSTACK(C,y) is followed, then the result is:

STRIPS System Running

$\mathrm{CLEAR}(y)$

$\mathrm{HOLD}(C)$

$\mathrm{CLEAR}(y) \wedge \mathrm{HOLD}(C)$

$\mathbf{STACK(C, y)}$

$\mathrm{CLEAR}(C)$

$\mathrm{ARMEMPTY}$

$\mathrm{ON}(C, y) \wedge \mathrm{CLEAR}(C) \wedge \mathrm{ARMEMPTY}$

$\mathbf{UNSTACK(C, y)}$ /* satisface HOLD(C) */

$\mathrm{CLEAR}(A) \wedge \mathrm{HOLD}(C)$

$\mathbf{STACK(C, A)}$

$\mathrm{ON}(B, D)$

$\mathrm{ON}(C, A) \wedge \mathrm{ON}(B, D) \wedge \mathrm{OTAD}$

STRIPS System Running

- A circular satisfaction of goals appears, in this case for the goal HOLD(C)
- This path is quitted
- The first alternative is chosen

STRIPS System Running

ONTABLE(C)

CLEAR(C)

ARMEMPTY

$\text{ONTABLE(C)} \wedge \text{CLEAR(C)} \wedge \text{ARMEMPTY}$

PICKUP(C) /* satisface HOLD(C) */

$\text{CLEAR(A)} \wedge \text{HOLD(C)}$

STACK(C, A)

ON(B, D)

$\text{ON(C, A)} \wedge \text{ON(B, D)} \wedge \text{OTAD}.$

STRIPS System Running

- In the state S_1 the goals ONTABLE(C) and CLEAR(C) are true, but ARMEMPTY is not true, due to the axiom:

$$(\exists x)\ (\mathrm{HOLD}(x)) \rightarrow \sim \mathrm{ARMEMPTY}$$

STRIPS System Running

- To satisfy the goal ARMEMPTY, there are two possible operators: STACK(x,y) and PUTDOWN(x)
- By applying the heuristic of the distance from the initial goal, STACK(B,D) is chosen
- There are also possible other instantitions for x and y, but the precondition HOLD(x) of the goal STACK(x,y) is satisfied only for x=B and the instance y=D is the closest to the initial goal

STRIPS System Running

$\mathrm{CLEAR}(D)$

$\mathrm{HOLD}(B)$

$\mathrm{CLEAR}(D) \wedge \mathrm{HOLD}(B)$

$\mathbf{STACK(B, D)}$

$\mathrm{ONTABLE}(C) \wedge \mathrm{CLEAR}(C) \wedge \mathrm{ARMEMPTY}$

$\mathbf{PICKUP(C)}$

$\mathrm{CLEAR}(A) \wedge \mathrm{HOLD}(C)$

$\mathbf{STACK(C, A)}$

$\mathrm{ON}(B, D)$

$\mathrm{ON}(C, A) \wedge \mathrm{ON}(B, D) \wedge \mathrm{OTAD}$

STRIPS System Running

- The preconditions of the operator STACK(B,D) are satisfied in the state S_1, so the operator can be applied and the transition to the state S_2 is done

$$S_1 \rightarrow S_2\text{: ONTABLE}(A) \wedge \text{ONTABLE}(C) \wedge \text{ONTABLE}(D) \wedge \text{ON}(B,D) \wedge \text{ARMEMPTY}$$

$$\text{Plan} = (\text{UNSTACK}(B,A), \text{STACK}(B,D))$$

STRIPS System Running

- The process continues and the final state S_4 is discovered and the plan which allows to obtain this state:

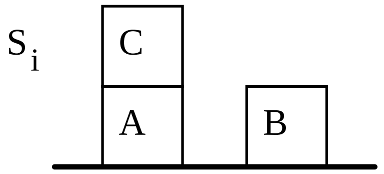
S_4: $\mathrm{ONTABLE}(A) \wedge \mathrm{ONTABLE}(D) \wedge \mathrm{ON}(B,D) \wedge \mathrm{ON}(C,A) \wedge \mathrm{ARMEMPTY}$

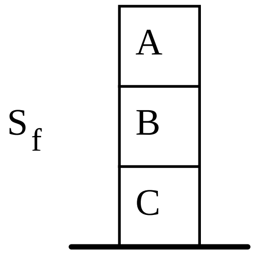
$\mathrm{Plan} = (\mathrm{UNSTACK}(B,A), \mathrm{STACK}(B,D), \mathrm{PICKUP}(C), \mathrm{STACK}(C,A))$

Sussman Anomaly

- Consider another instance of the planning problem in the blocks world, known as the Sussman anomaly

Sussman Anomaly





ON(C,A) $\wedge$
ONTABLE(A) $\wedge$
ONTABLE(B) $\wedge$
ARMEMPTY

ON(A,B) $\wedge$
ON(B,C)

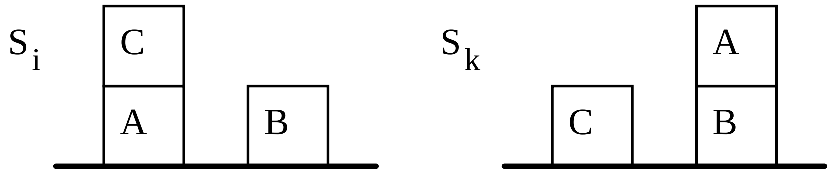
Stack 1	Stack 2
ON(A,B)	ON(B,C)
ON(B,C)	ON(A,B)
ON(A,B) $\wedge$ ON(B,C)	ON(A,B) $\wedge$ ON(B,C)

Sussman Anomaly

- Consider solving this problem with the STRIPS planning system
- The goals to be satisfied for reaching the final state generate two stacks
- Starting with the goals from the Stack 1, a sequence of operators is generated, which determines the transition from the initial state S_i in the state S_k

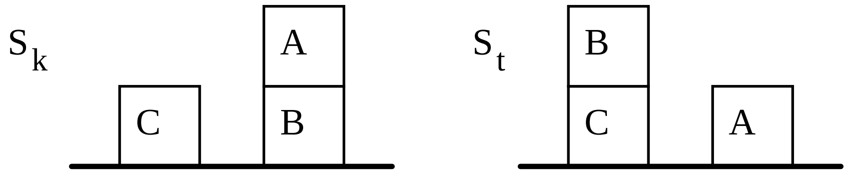
Sussman Anomaly

Stack 1



$S_i \rightarrow S_k$: ONTABLE(B) $\wedge$ ON(A,B) $\wedge$ ONTABLE(C) $\wedge$ ARMEMPTY

$\text{Plan}_{S_i \rightarrow S_k}$ = (UNSTACK(C,A), PUTDOWN(C), PICKUP(A), STACK(A,B))



$S_k \rightarrow S_t$: ON(B,C) $\wedge$ ONTABLE(A) $\wedge$ ONTABLE(C) $\wedge$ ARMEMPTY

$\text{Plan}_{S_k \rightarrow S_t}$ = (UNSTACK(A,B), PUTDOWN(A), PICKUP(B), STACK(B,C))

$S_t \rightarrow S_f$: ON(A,B) $\wedge$ ON(B,C)

$\text{Plan}_{S_t \rightarrow S_f}$ = (PICKUP(A), STACK(A,B))

Plan = (UNSTACK(C,A), PUTDOWN(C), PICKUP(A), STACK(A,B), UNSTACK(A,B)
PUTDOWN(A), PICKUP(B), STACK(B,C), PICKUP(A), STACK(A,B))

Sussman Anomaly

- In the state S_k the goal ON(A,B) is satisfied, it is eliminated from the stack and the goal ON(B,C) will start to be fulfilled
- To satisfy this goal, the block A must be put on the table, in order to move the block B over thr block C
- When the goal ON(B,C) is satisfied, the operators which determine the transition from the state S_k into the state S_t were applied

Sussman Anomaly

- Because the goal ON(B,C) is satisfied in the state S_t it is eliminated from the stack
- ON(A,B) $\wedge$ ON(B,C) remains in the stack
- This goal is not satisfied, although the goal ON(A,B) was initially satisfied
- The goal was invalidated due to the fulfillment of the goal ON(B,C)
- The difference between the current state and the goal state is now ON(A,B)

Sussman Anomaly

- The goal ON(A,B) is introduced in the stack and is satisfied, by applying two operators, which determines the transition from S_t to S_f
- Although the plan is successful, it is not efficient
- A similar situation appears when solving the problem using the Stack 2
- STRIPS planning system can not find an efficient solution for this problem

Sussman Anomaly

- The difficulty is generated by the fact that there is no combinations of plans which, solving separately the two subgoals, solve also the conjunction of the subgoals
- In order to eliminate the Sussman anomaly, it is necessary that the planner will consider, simultaneously, many goals which interact, so it will execute a non-linear planning

Non-Linear Planning

- Sussman anomaly shows that the fulfillment of conjunction of goals which interact create problems in the plan synthesis
- For certain problems, it is not possible to find an order for the conjunction of goals, such that the plans for satisfying some goals will not interfere with the plans for satisfying the other goals

Non-Linear Planning

- An efficient plan for solving the Sussman anomaly is:
  - 1. Satisfy the goal by making free the block A and placing the block C on the table
  - 2. Satisfy the goal by placing the block B over the block C
  - 3. Perform the rest of the operations by satisfying the goal by placing the block A over the block B

Non-Linear Planning

- This plan can be synthesized by a non-linear planning method
- While a plan is built, it is not necessary to specify completely the execution order for the operations
- A plan specifies only a partial order for the operations, at least until the moment the plan is complete

Non-Linear Planning -TWEAK

- The non-linear planning system TWEAK is domain independent and uses as a planning technique, the constraint recording
- **Constraint recording** is the process of defining an object (a plan in this case), by incrementally specifying the partial descriptions (constraints) which the object must fulfill

Non-Linear Planning -TWEAK

- Constraint recording can be seen as a search strategy in which parts of the search space are successively eliminated by adding constraints
- Constraints exclude states and search paths until all the remaining alternatives are satisfactory, at the end

Non-Linear Planning -TWEAK

- The advantage of constraints recording is the fact that the properties of the searched object (the plan) must not be chosen until the moment in which there is a reason for choosing them
- The constraints recording is a delayed decision strategy problem solving paradigm

Plan Representation

- A plan developed by the TWEAK system is a sequence of plan steps (operators, actions)
- A **step plan** is represented by the action which must be executed, a finite set of **preconditions** (equivalent with the preconditions list from STRIPS) and a finite set of **postconditions**

Plan Representation

- The set of the effects of an action, that is postconditions, is formed by AL and EL
- Preconditions represent assertions about the problem current state, which must be true such that the action can be executed
- Postconditions represent assertions validated and invalidated as an effect of executing the plan step

Plan Representation

- Preconditions and postconditions are expressed by atomic formulas in first order predicate logic, which accepts as arguments constants and variables
- The functional symbols, the operators and the quantifiers are not allowed in preconditions and postconditions

Plan Representation

- The postconditions corresponding to the assertions validated by an action are represented by atomic formulas, and those corresponding to invalidated assertions, by negated atomic formulas
- The representation power of the TWEAK system is equivalent to the STRIPS system

Operators Representation

- Considering the blocks world and the previously defined plan operators, a representation of two of these operators is:

Operators Representation

Action	$STACK(x,y)$
Preconditions	$CLEAR(y) \wedge HOLD(x)$
Postconditions	$ARMEMPTY \wedge ON(x,y) \wedge \sim CLEAR(y) \wedge \sim HOLD(x)$

Action	PICKUP(x)
Preconditions	$CLEAR(x) \wedge ONTABLE(x) \wedge ARMEMPTY$
Postconditions	$HOLD(x) \wedge \sim ONTABLE(x) \wedge \sim ARMEMPTY$

Planning Activity

- During the planning activity, the problem state is represented by a set of formulas
- While the plan steps are executed, the problem state is modified
- The representation is similar to those used by STRIPS system

Planning Activity

- A plan has an associated inital state and a final state
- In a similar manner, for each plan step, the input and the output states are defined, as being the sets of formulas which are true at the beginning and respectively at the end of executing a plan step

Planning Activity

- TWEAK system uses plans in which the operators execution order is not completely specified
- During the problem solving process, the TWEAK system develops an incomplete plan
- An **incomplete plan** is a partial description of a plan capable to solve the proposed problem

Planning Activity

- In each moment, TWEAK system has a set of useful plan steps, but it does not have a clear idea about the way in which it will order, in the end, this set of actions for executing the plan
- The plans will be represented by sets of plan steps, considering that the steps order is not yet established:

Plan = {STACK(A,B), PICKUP(B), PICKUP(A)}

Planning Activity

- The partial plan can be completed, so transformed in a complete plan, in different ways, by adding constraints in the partial plan
- An incomplete plan represents a class of complete plans
- The planning process consists in adding new constraints to the incomplete plans and ends when a plan is obtained (complete or incomplete), which solves the proposed problem

Planning Activity

- In the TWEAK system, the plans can be incomplete due to:
  - Incomplete specification of the temporal order of plan steps execution
  - Incomplete specification of plan steps

Planning Activity

- Generating a complete plan from an incomplete one can be done by adding two types of constraints:
  - **Temporal constraints** which specify the execution order of the plan steps
  - **Substitution constraints** which specify completely the plan steps

Planning Activity

- A temporal constraint is represented by conditioning the execution of a plan step before another plan step
- A set of temporal constraints represent a partial order for the plan steps

Planning Activity

- A complete plan represents a total order of a finite set of plan steps
- The order relation is the temporal order and the plan steps are represented by their associated actions
- Plan fulfillement corresponds to the execution of the actions associated to the plan steps, in the order established by the plan

Planning Activity

- The substitution constraint is an equivalence relation between the variables and the constants from the formulas
- In a complete plan, all the variables which appear in preconditions and postconditions must be linked to a certain constant
- When executing the actions associated to the plan steps, the variable which appear in a plan step will be substituted by the constant

Planning Activity

- The substitution constraints impose the substitution of the elements from the formulas
- The substitution of the formulas is similar to the unification of formulas from first order predicate logic

Planning Activity

- Adding a constraint to an incomplete plan can destroy all the complete extensions of the incomplete plan
- In this case, the set of constraints is inconsistent and does not represent any more a valid incomplete plan
- The system must perform a backtracking procedure to choose another constraint to be added in the incomplete plan

Plan Synthesis

- The goal of the TWEAK system is to build a plan for a given problem
- The sythetized plan must contain the actions which transform the initial state into the final state, so it must validate all the formulas from the problem final state
- Initially, an empty plan is created, without any step

Plan Synthesis

- While the planning process advances, a goal (formula) which must be satisfied is chosen and the plan steps are added, incompletely specified, both as instantiations, and as temporal order, forming an incomplete plan
- Satisfying each goal determines the gradual refinement of the incomplete plan, based on the plan modification operations
- The plan which solves the problem can be both complete, as incomplete, case in which every instance of the incomplete plan can produce the solution

Plan Synthesis

- In the TWEAK system there are five operations for plan changing:
- (1) **Adding steps** – operation through which new steps are created and are added to the plan
- (2) **Promotion** – operation to establish a temporal order between two plan steps

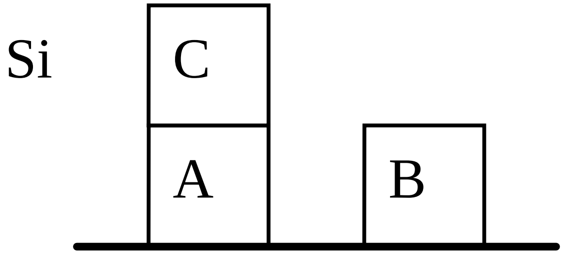
Plan Synthesis

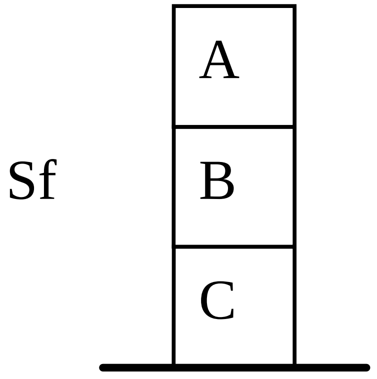
- (3) **Simple linking** – operation to assign values to variables in order to validate the preconditions of a plan step
- (4) **Separation** – operation to block the assignment of certain values to a variable
- (5) **Destroy elimination** – operation of introducing a new step S_3 (a step which already exists in the plan or a new step) between the steps S_1 and S_2, in order to add a fact invalidated by the step S_1 which is necessary in the step S_2

Plan Synthesis

- Solving the Sussman anomaly can be done using the TWEAK system
- Suppose that the system is capable to select each time the plan steps which conduct towards the solution
- In fact, the system will perform a backtracking procedure to choose the right alternative for completing the plan

Sussman Anomaly





$ON(C,A) \wedge$
$ONTABLE(A) \wedge$
$ONTABLE(B) \wedge$
$ARMEMPTY$

$ON(A,B) \wedge$
$ON(B,C)$

Sussman Anomaly

- The goal state is:
- $ON(A,B) \wedge ON(B,C)$
- The planning process starts with an empty plan
- Based on the means end analysis principle, in order to reach the goal state, two plan steps are chosen, which contain in the postconditions list one of the two goals

Sussman Anomaly

- Each operator is represented with the preconditions list above the operator and the postconditions list below the operator
- The invalidated assertions are marked in front of them by the symbol ~
- A precondition which is not fulfilled in the current state is marked by the symbol *
- To satisfy the two assertions from the final state, the following steps can be applied and the plan is modified by adding these two steps:

Sussman Anomaly

CLEAR(B)	CLEAR(C)
*HOLD(A)	*HOLD(B)
------------------	------------------
STACK(A,B)	STACK(B,C)
------------------	------------------
ARMEMPTY	ARMEMPTY
ON(A,B)	ON(B,C)
~ CLEAR(B)	~ CLEAR(C)
~ HOLD(A)	~ HOLD(B)

Plan = {STACK(A,B),STACK(B,C)}

Sussman Anomaly

- The plan is modified by adding new operators which allows to satisfy the preconditions of the two plan steps
- The heuristic used to introduce new plan steps for precondition satisfaction corresponds to the plan changing operation of **adding steps**
- Because CLEAR(B) and CLEAR(C) are already satisfied in the initial state, the goals which must be satisfied, by changing the plan, are HOLD(A) and HOLD(B)
- The following steps are added to the incomplete plan:

Sussman Anomaly

*CLEAR(A)	*CLEAR(B)
ONTABLE(A)	ONTABLE(B)
*ARMEMPTY	*ARMEMPTY
-------------------	---------------
PICKUP(A)	PICKUP(B)
-------------------	---------------
~ ONTABLE(A)	~ ONTABLE(B)
~ARMEMPTY	~ ARMEMPTY
HOLD(A)	HOLD(B)

Sussman Anomaly

- In this moment, the incomplete plan contains the unordered set of steps:

Plan = {STACK(A,B),STACK(B,C),PICKUP(A),PICKUP(B)}

Sussman Anomaly

- If a plan step PICKUP follows the STACK step in the final plan, the goals HOLD(A) and HOLD(B) should be satisfied in another way and they can not be satisfied by the PICKUP step
- This problem is solved by introducing temporal constraints for ordering the plan steps, of the form $P_1 < P_2$, meaning that: the plan step P_1 must precede the plan step P_2 in the final plan

Sussman Anomaly

- The temporal restrictions are:
- PICKUP(A) < STACK(A,B)
- PICKUP(B) < STACK(B,C)
- The heuristic used for ordering the steps corresponds to the plan changing operation called **promotion**

Sussman Anomaly

- The incomplete plan contains four steps partially ordered and four unsatisfied preconditions
- The formula CLEAR(A) is unsatisfied, because the block A is not free in the initial state
- The formula CLEAR(B) is unsatisfied because, although the block B is free in the initial state, there is the plan step STACK(A,B), having the postcondition ~CLEAR(B), and this step can precede the plan step which has the formula CLEAR(B) as precondition, that is the plan step PICKUP(B)

Sussman Anomaly

- In order to prevent this, another ordering constraint is introduced:
- PICKUP(B) < STACK(A,B)
- There are still unsatisfied goals ARMEMPTY (for both PICKUP operators) and CLEAR(A)
- Although in the initial state the ARMEMPTY precondition is satisfied, each PICKUP operation has as effect ~ARMEMPTY

Sussman Anomaly

- Each executed PICKUP operation can make impossible the execution of the other PICKUP operation
- Ordering is used to fulfill at least a PICKUP step:
- PICKUP(B) < PICKUP(A)

Sussman Anomaly

- Because in the initial state the robot arm is free and no plan step, which precedes the PICKUP(B) step, does not invalidate the goal ARMEMPTY, the preconditions of the PICKUP(B) step are satisfied
- To fulfill the precondition ARMEMPTY of the step PICKUP(A), a third heuristic is used, corresponding to the plan changing operation called **destroy elimination**

Sussman Anomaly

- The step PICKUP(B) has as effect the invalidation of the goal ARMEMPTY, that is ~ARMEMPTY, but if it is possible to introduce another plan step between the steps PICKUP(B) and PICKUP(A), which will validate again the goal ARMEMPTY, then the precondition of the PICKUP(A) step will be satisfied

Sussman Anomaly

- This can be done using the STACK(B,C) step, which is temporally constrained as:
- PICKUP(B) < STACK(B,C) < PICKUP(A)

Sussman Anomaly

- It is found a situation in which the PICKUP(B) step destroys the preconditions of the PICKUP(A) step, but the STACK(B,C) step eliminates these destructions
- In this moment, the only precondition which must still be fulfilled is CLEAR(A)
- To satisfy this condition, a new plan step is added:

Sussman Anomaly

*ON(x,A)

*CLEAR(x)

*ARMEMPTY

UNSTACK(x,A)

~ARMEMPTY

CLEAR(A)

HOLD(x)

~ ON(x,A)

Sussman Anomaly

- The temporal constraints and the plan become:

PICKUP(B) < STACK(B,C) < PICKUP(A) < STACK(A,B)

Plan = {STACK(A,B),STACK(B,C),PICKUP(A),PICKUP (B),

STACK(B,C),UNSTACK(x,A)}

Sussman Anomaly

- The variable x is introduced, because the only postcondition which is important is CLEAR(A) and it is not important which block is over the block A
- The ordering constraints allow to incompletely specify the plans from the point of view of step ordering and the variables allow the incomplete specification of the plans, by partially instantiating the steps (operators)

Sussman Anomaly

- After the step UNSTACK(x,A) is introduced, there are three unsatisfied preconditions
- The first one, ON(x,A), can be satisfied by the instantiation constraint of x to C, that is a substitution constraint
- This corresponds to the plan changing operation called **simple linking**
- In this moment, the preconditions become ON(C,A), CLEAR(C) and ARMEMPTY

Sussman Anomaly

- The preconditions CLEAR(C) and ARMEMPTY can be invalidated by plan steps, but this is blocked by **promotion** plan changing operations:
- UNSTACK(x,A) < STACK(B,C)
- UNSTACK(x,A) < PICKUP(A)
- UNSTACK(x,A) < PICKUP(B)

Sussman Anomaly

- But for the step PICKUP(B) it is necessary to fulfill the precondition ARMEMPTY, which is invalidated by the new added step UNSTACK(x,A)
- Solving this problem can be done by adding a new step for destruction elimination

Sussman Anomaly

HOLD(C)

PUTDOWN(C)

~ HOLD(C)
ONTABLE(x)
ARMEMPTY

Sussman Anomaly

- It is necessary the ordering:

$$\text{UNSTACK}(x, A) < \text{PUTDOWN}(C) < \text{PICKUP}(B).$$

Sussman Anomaly

- Two ways for destruction elimination are shown: one in which a step which already exists, STACK(B,C), is used, and another one, in which it is necessary to introduce a new step, that is PUTDOWN(C)
- The preconditions of the PUTDOWN step are satisfied, so the preconditions of all the plan steps are satisfied
- The complete plan is:

Sussman Anomaly

Plan = {1. UNSTACK (C,A)
2. PUTDOWN (C)
3. PICKUP (B)
4. STACK (B,C)
5. PICKUP (A)
6. STACK (A,B) }

Conclusions

- The non-linear planning method from the TWEAK system generates an efficient plan for solving the Sussman anomaly, that is a problem in which the subgoals satisfaction interact