Hierarchical Planning

Main problem in STRIPS-like planning (as well as in other planning frameworks): **complexity**

One reason for complexity: **no structure**

- no distinction between important and unimportant properties
- no distinction between important and unimportant operators

This observation gives raise to two different ways of **abstraction** in planning:

- **abstraction of situations** and
- **abstraction of operators**

Abstraction of Situations

ABSTRIPS (**Abstraction-Based** STRIPS)

Main idea: introduce **weights** for each literal and consider only the most important ones in first loop, then refine by considering also literals with second highest weight.

In blocksworld, for instance, properties with weights:

Property	Weight
`On`	4
`Clear`	3
`Holds`	2
`Ontable`	2
`Handempty`	1

Higher weights indicate more important properties. Means here concretely: first consider only the property `On`, in the second loop the properties `On` and `Clear` and so on. Use the abstract plan for a refinement of the more detailed plans. In the last loop all properties have to be considered.

Operators of the Blocksworld

PICKUP(x)		PUTDOWN(x)	
preconditions	Clear[3](x)	**preconditions**	Holds[2](x)
	Ontable[2](x)	**delete list**	Holds(x)
	Handempty[1]	**add list**	Clear(x)
delete list	Clear(x)		Ontable(x)
	Ontable(x)		Handempty
	Handempty		
add list	Holds(x)		
STACK(x, y)		UNSTACK(x, y)	
preconditions	Holds[2](x)	**preconditions**	Clear[3](x)
	Clear[3](y)		On[4](x, y)
delete list	Holds(x)		Handempty[1]
	Clear(y)	**delete list**	Clear(x)
add list	Clear(x)		On(x, y)
	On(x, y)		Handempty
	Handempty	**add list**	Holds(x)
			Clear(y)

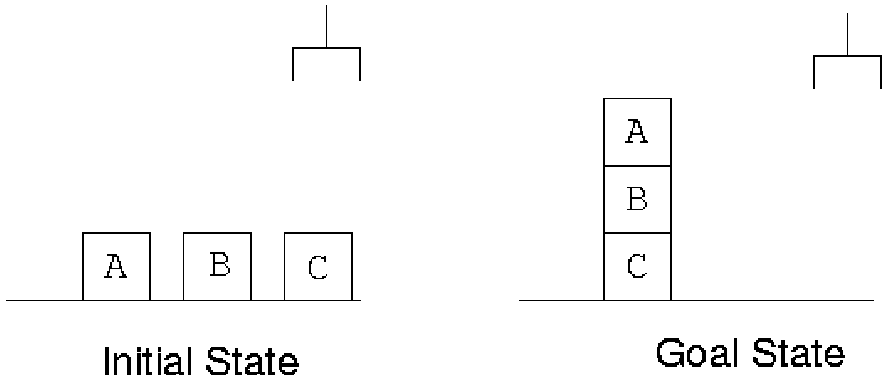
Non-Linear Planning with Abstraction of Situations

Assume non-linear planning algorithm NLP with input: partial non-linear plan & output: total well-formed conflict-free non-linear plan. NLPAS (Non-Linear Planning with Abstraction of Situations) calls NLP initially considering only the most important properties. This plan is refined by taking into account less important properties.
Input: non-linear plan and index (initially the highest weight)

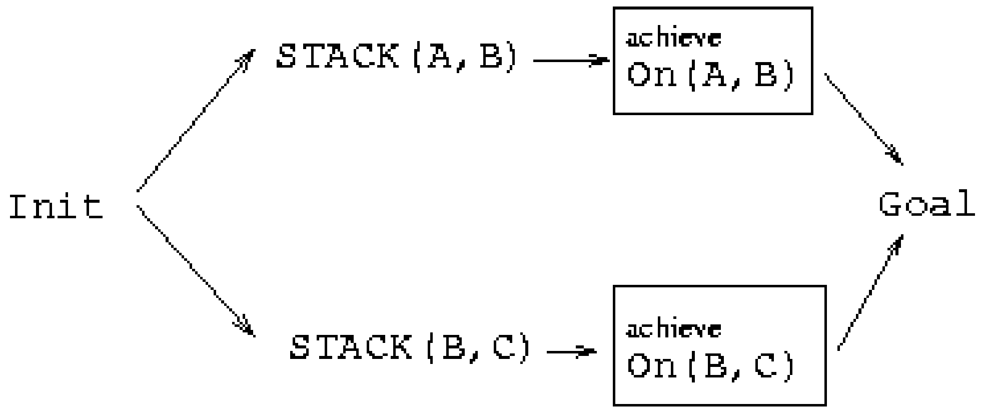
```
begin
 0.   if Index=0 then return(P)
 1.   else
 1.1      disregard all preconditions of operators,
          whose weights are smaller than Index
 1.2.     select P':=NLP(P)   ;;; arbitrary choice → backtrack point
      endif
 1.3.   return(NLPAS(P',Index-1))
end
```

Example

Consider the following easy planning problem:

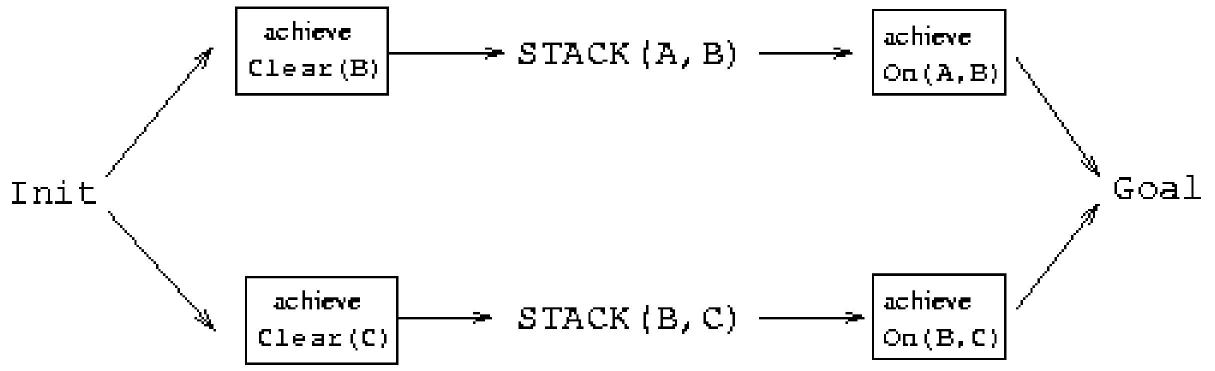


Result of NLPAS with weight 4 (consider only On):

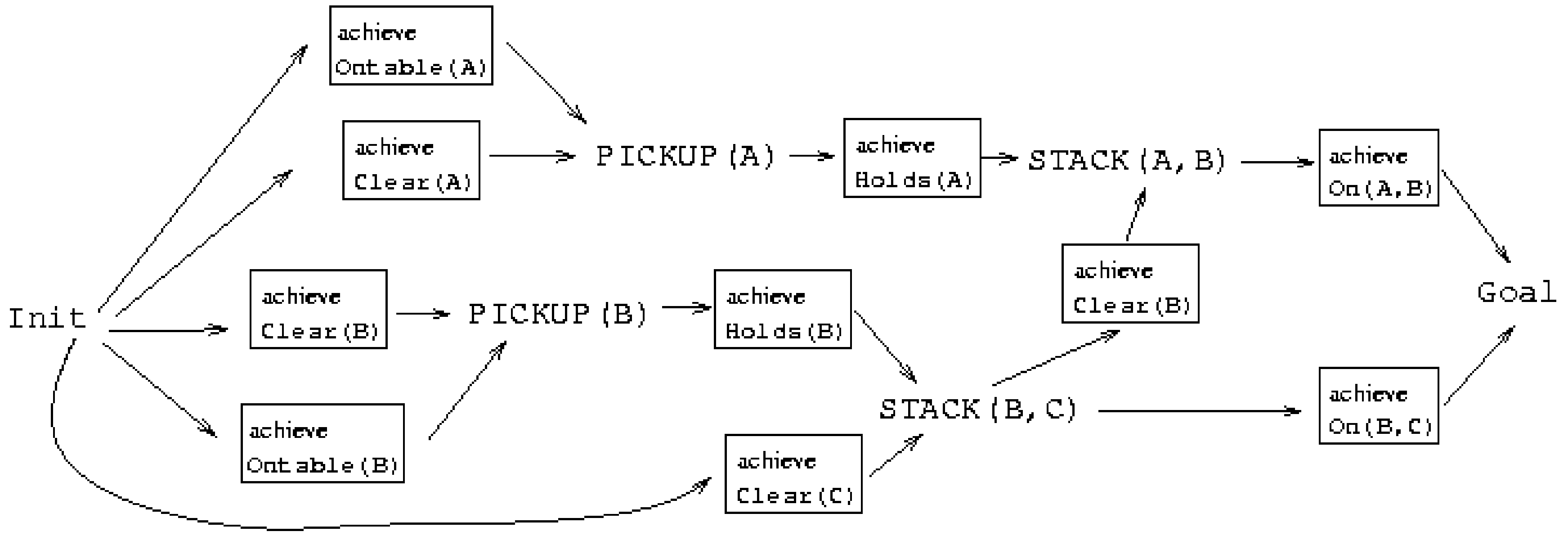


Example (Cont'd)

Result with weight 3 (consider `Clear` in addition):

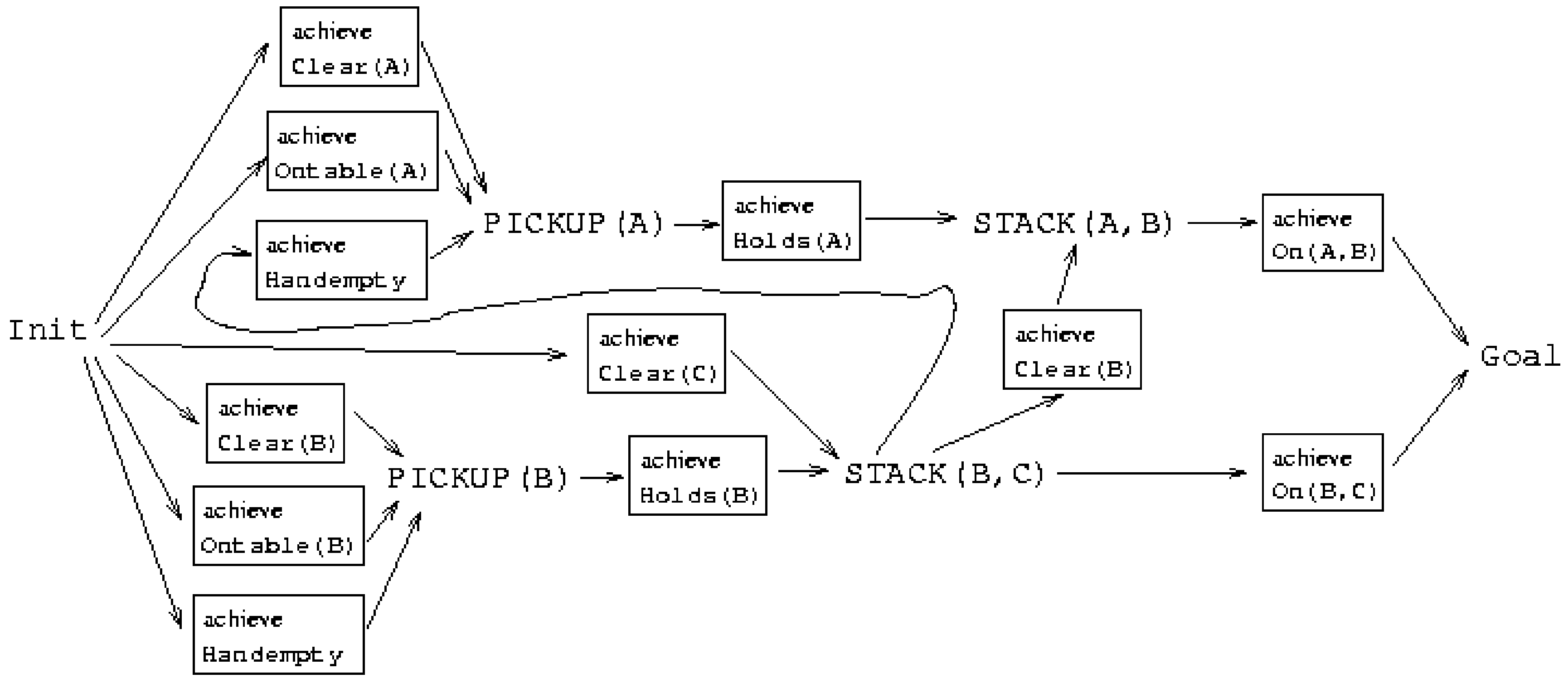


Result with weight 2 (consider `Holds` and `Ontable` in addition):



Example (Cont'd)

Final result, i.e. weight 1 (consider `Handempty` in addition):



Problems with Abstraction

- Abstraction certainly plays an **important role in human problem solving**. Questionable whether the proposed approach is adequate.
- **Selection of a useful order**. For instance, is the order given above useful or not?
- For achieving the requirements of achieve goals it is necessary to **add new nodes. Which ones?** Can sometimes be seen only on more concrete level. E.g. achieve `Holds(A)` can be achieved by `UNSTACK(A,x)` and `PICKUP(A)`, which one is better depends on situation of `A`. If `A` on the table then `PICKUP(A)`, but on abstract level `Ontable` is invisible.

What to do?

- accept **inefficient plans**.
- **adapt the non-linear planning algorithm** such that conflicts are only resolved on more concrete levels.

Discussion

- **Good decisions on the higher levels are even more important** than in non-hierarchical planning. In particular deviations should be avoided (since they might have to be expanded).
- There are classes of examples which become **tractable** by using hierarchical planning instead of non-hierarchical planning.
- There are classes of examples which become **intractable** by using hierarchical planning instead of non-hierarchical planning.

Abstraction of Operators

Look at motion planning for a robot

Question: On which level should the operators be given?

Example blocks-world: four operators
`PICKUP`, `PUTDOWN`, `STACK`, and `UNSTACK`

More detailed operators to split `PICKUP` into three parts:

- `POSITION` for positioning the robot's hand
- `CATCH` for grasping it
- `LIFT` to lift it.

Give specification for these! Operator `PICKUP` can be viewed as an **operator abstraction** of the operators `POSITION`, `CATCH`, and `LIFT`

An Abstract Operator

Define operator `FREE` as common abstraction of `UNSTACK` and `PUTDOWN`

`FREE(x, y)`	
preconditions	`Handempty`
	`On(x, y)`
delete list	`On(x, y)`
add list	`Clear(y)`
plot	`(UNSTACK(x, y), PUTDOWN(x))`

It differs from the other operators in the blocksworld that **not all preconditions are deleted**.

preconditions of `FREE` that robot's hand empty, but is empty after the execution too

New entry **plot**, which tells that planning has to go on after abstract planning.

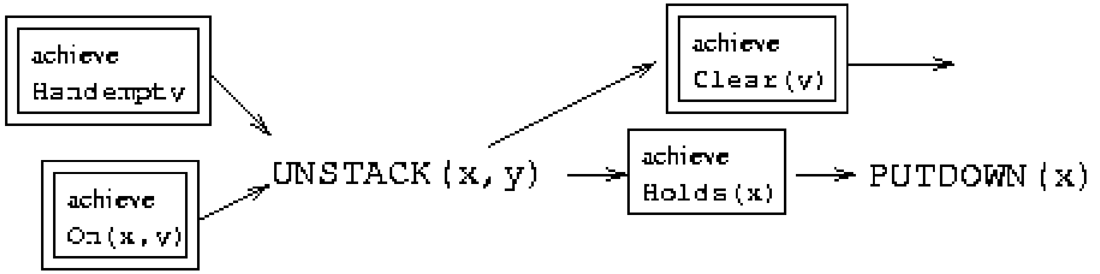
Operator has to be adjusted to a particular planning data structure, here list of STRIPS

The same for non-linear planning

FREE(x, y)

preconditions	Handempty
	On(x, y)
delete list	On(x, y)
add list	Clear(y)

plot



The dependencies which go into the plan and leave it are marked by preconditions and add list.

Important property of abstract planning: preconditions, add list, **and** delete list **need not to be precisely characterised**

E.g. it is not specified that after the application of FREE(x,y) always Ontable(x)

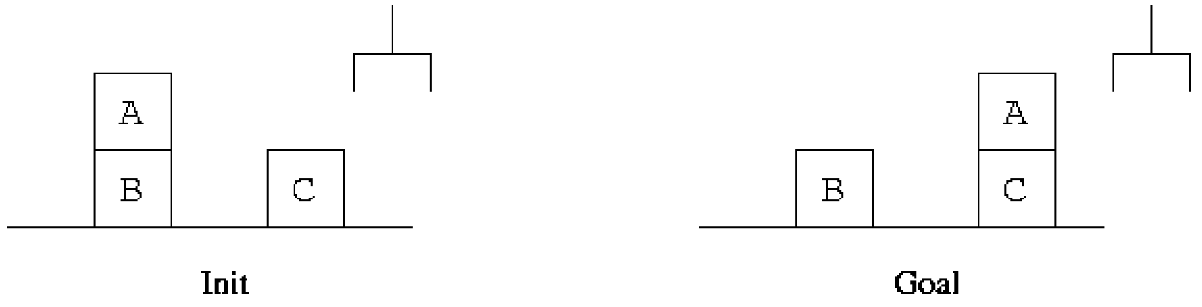
Non-optimality of Abstraction

Main Question: **Which degree of abstraction?**

Find compromise between

- **disregarding unimportant details** (e.g. side-effect of `FREE(A,B)` that always `Ontable(A)`)
- **considering details in order to find an optimal plan** (e.g. if for getting `Clear(B)` in an abstract plan `FREE(A,B)` is applied and we want in addition `Ontable(A)`, then we have to plan for `Ontable(A)` too)

Example



Ontable(B)	Clear(C)	On(A,C)	Clear(B)
On(A,B)	Handempty	Ontable(C)	Handempty
Clear(A)		Clear(A)	

Solve with operators STACK, UNSTACK, PICKUP, PUTDOWN, and FREE

In order to achieve Clear(B), select FREE(A,B), hence A has to be put on C in the next step. Linearised plan: (FREE(A,B),PICKUP(A),STACK(A,C)) after expansion of FREE(A,B):
(UNSTACK(A,B),PUTDOWN(A),PICKUP(A),STACK(A,C))

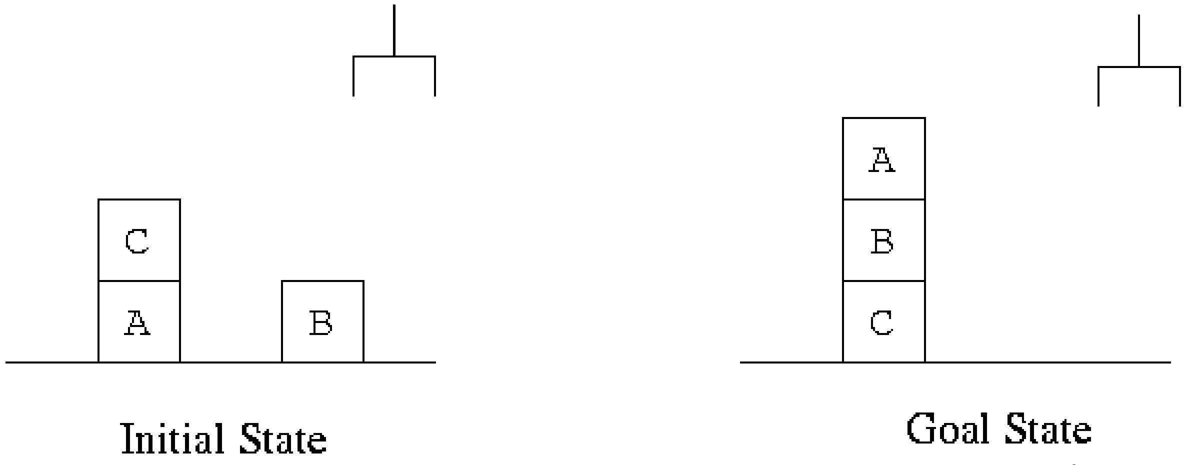
this plan contains deviation (PUTDOWN(A),PICKUP(A))

Pseudo code for abstract non-linear planning

input: initial non-linear plan output: non-linear plan

```
begin
 1.    while  nodes unsolved or unexpanded in P do
 1.1.     select unsolved or unexpanded node N from P
          ;;; arbitrary choice → backtrack point
 1.2 if N  unsolved
       then
 1.2.1    select action A, which has N
          in add list   ;;; arbitrary choice → backtrack point
 1.2.2    insert A as operator node immediately
          before N in P  add achieve-nodes for the preconditions of A
          in parallel immediately before A
       else
 1.2.3    replace N by its plot and supplement preconditions
       end_if
 1.3.     resolve interactions
 1.4.     criticise plan
       end_do
 2.    return(P)
end
```

Example: Sussman Anomaly

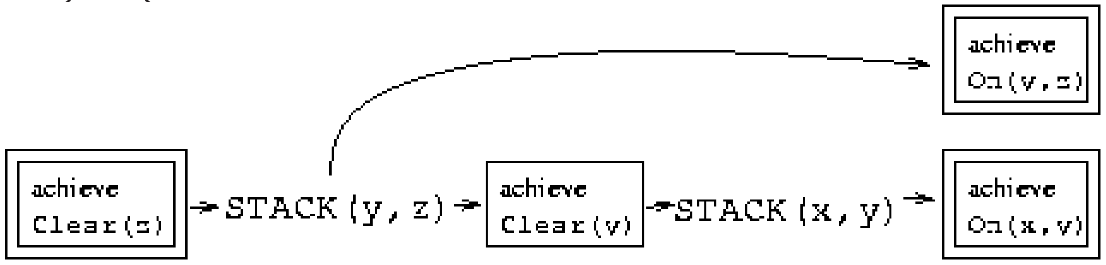


Operators PICKUP, PUTDOWN, STACK, UNSTACK, FREE, and STACKTHREE (for stacking three blocks)

STACKTHREE(x, y, z)

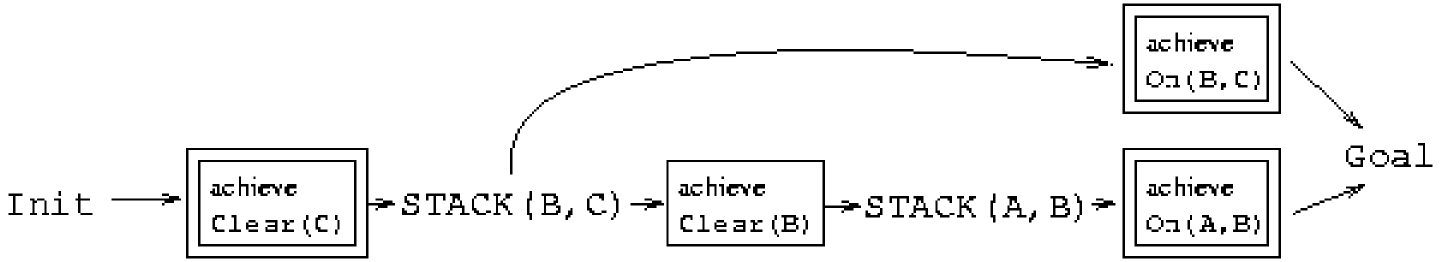
preconditions	Clear(z)
delete list	Clear(z)
add list	On(x, y)

plot

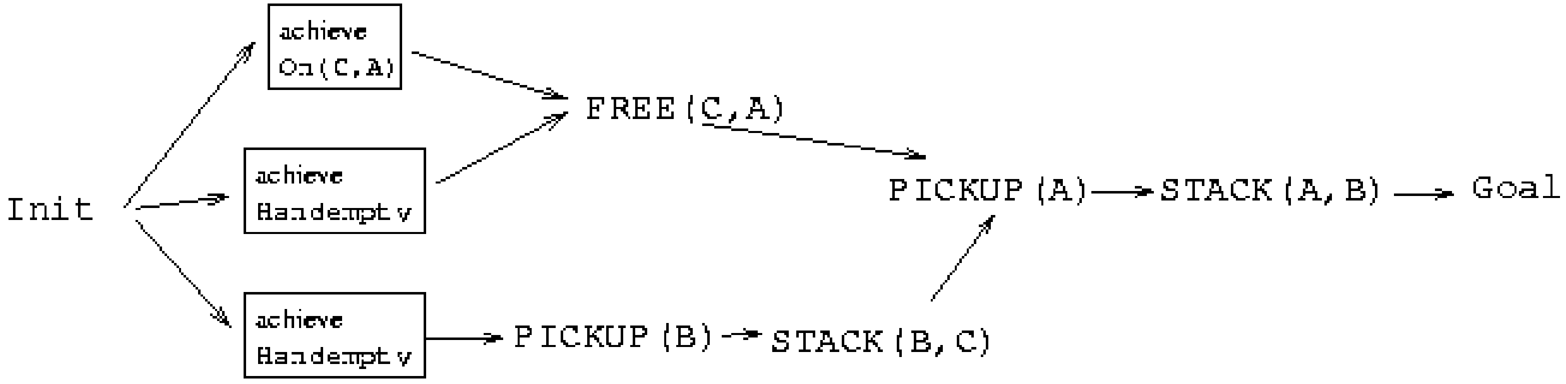


Example (Cont'd)

In order to solve the problem, simply use operator `STACKTHREE(A,B,C)`. Solves goal since `On(A,B)` and `On(B,C)` in **add list**. Applicable because of `Clear(C)`. Next loop: expansion of the abstract operator, i.e.



Then look at preconditions of expanded operators:



Example (Cont'd)

Conflict between the operators
`FREE(C,A)` and `PICKUP(B)`
with respect to the property `Handempty`.

How to solve: Linearisation: order `FREE(C,A)` before or after the parallel planning part
In this case useful to bring the `FREE(C,A)`-operator in front, but not possible to decide since the precondition `Clear(C)` is not seen.

Alternative: Resolve conflicts after expansion

Discussion

- **New:** expansion of nodes.
- **Replacement of abstract nodes makes explanation and backtracking harder.**
- **No distinction between abstract planning phase and expansion phase**
- abstract operators can be **extraordinary useful**, in particular necessary to model high-level intelligent behaviour.
- Abstraction in planning adds **many additional problems** and can in no way considered as solved.

Literature

- E.D. Sacerdoti. *A Structure for Plans and Behavior*. North Holland. 1997.
- D. Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, **32**:333–377, 1987.
- Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, **5**:115–135, 1974.
- Craig A. Knoblock. Search reduction in hierarchical problem solving, In AAAI-91, pp.686–691, 1991.
- Christer Bäckström and Peter Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. *IJCAI'95*, pp.1599–1604.