规划问题求解（AI Planning）

毛文吉

中国科学院自动化研究所

Planning agent

- Since the early 1970s [Fikes & Nilsson], the AI planning community has been closely concerned with the design of artificial agents
- AI planning system is a central component of any artificial agent
- Planning is essentially the automatic generation of a course of actions to achieve some desired goal
- Many planning algorithms have been proposed and planning has become a well-developed field in AI and agent research

What is planning

- An *automatic reasoning* process to generate plans of a sequence of actions for achieving certain goal(s)

Planning questions

Question 1: How do we *represent*. . .

- Goals to be achieved
- States of environment
- Actions available to agent
- Plans itself

Question 2: How do we use these representations to generate *plans*?

- What kind of reasoning should be involved?

Outline

- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

Outline

- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

Strips (Fikes and Nilsson 71)

- Highly influential representation for actions:
  - Preconditions: list of propositions to be true（前提条件表）
  - Delete list: list of propositions that will become false（删除表）
  - Add list: list of propositions that will become true（增加表）

- Example

  Initial state: at(home), ¬have(beer), ¬have(chips)

  Goal: have(beer), have(chips), at(home)

  Actions:

  Buy (X):
  Precond: at(store)
  Add: have(X)

  Go (X, Y):
  Precond: at(X)
  Delete: at(X)
  Add: at(Y)

Frame problem（框架问题）

- I go from home to the store, creating a new situation S’. In S’:
  - The store still sells chips
  - My age is still the same
  - Beijing is still the capital city of China…

- How can we efficiently represent everything that hasn’t changed?

Ramification problem（分支问题）

- I go from home to the store, creating a new situation S'. In S':
  - I am now in Zhongguancun
  - The number of people in the store went up by 1
  - The contents of my pockets are now in the store...
- Do we want to say all that in the action definition?

Strips treatment

In Strips, some facts are inferred within a world state

e.g. the number of people in the store

- Primitive facts, e.g. *at(home)* persist between states unless changed
- Inferred facts are not carried over and must be re-inferred
  - Avoids making mistakes, perhaps inefficient

Strips representation for actions:

Move-C-from-A-to-Table:

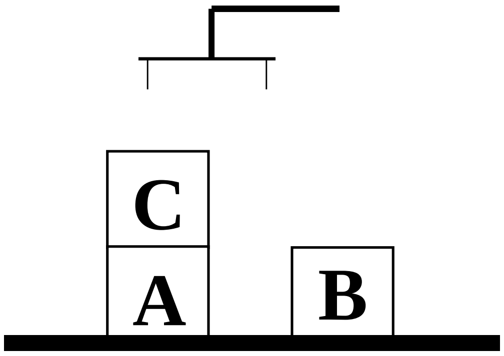
Precondition: on(C, A), clear(C)

Effect:

add: on-table(C)

clear(A)

delete: on(C, A)



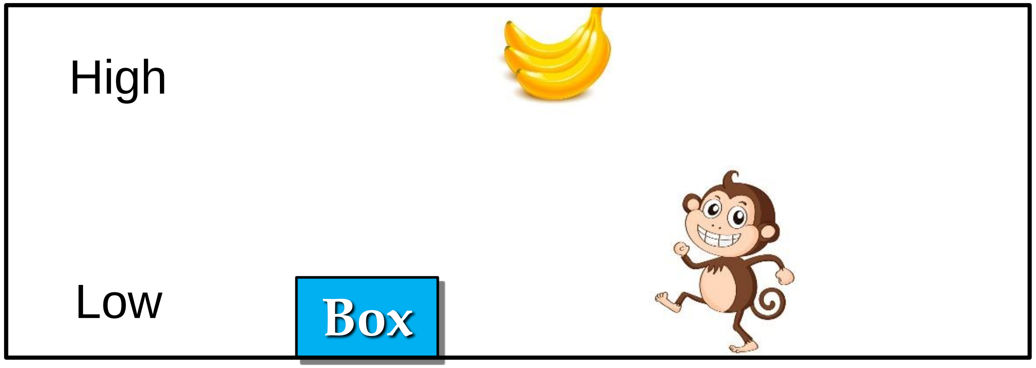
- The explicit effects are the only changes to the state

Means-ends analysis（手段目的分析）

Strips’s problem solving takes **means-ends analysis**

- Search by reducing the difference between state and goals
- What *means* (operators) are available to achieve the desired *ends* (goals)

The monkey and bananas problem（猴子香蕉问题）

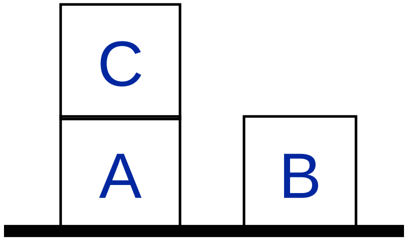


Blocks world example (Sussman anomaly)

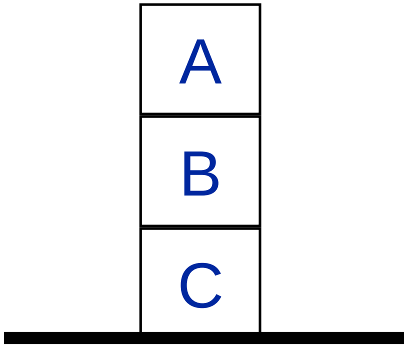
分成子任务. 把各自子任务完成.

- Strips uses *noninterleaved planner* (非交叉规划器), which cannot solve this example …

Initial:



Goal:



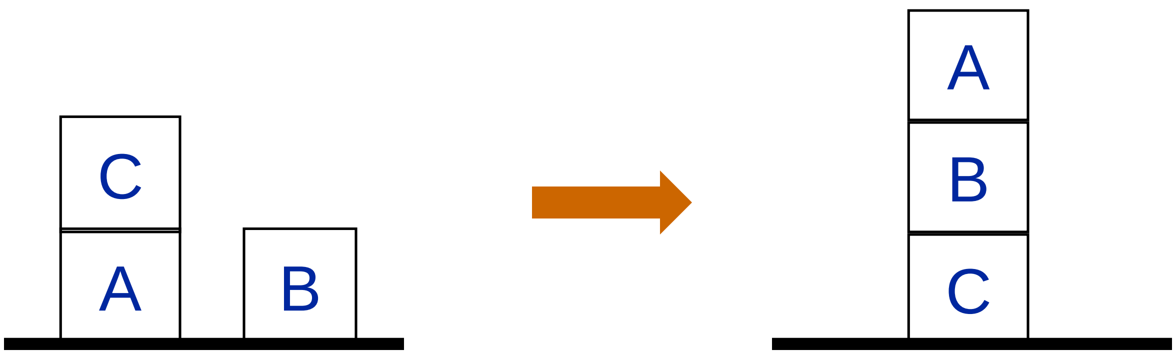
Initial state: on(C, A), on-table(A), on-table(B), clear(B), clear(C)

Exercise

- A noninterleaved planner is a planner that, when given two subgoals G1 and G2, produces either a plan for G1 concatenated with a plan for G2, or vice versa.

分解成子目标不一定完成总目标

Why a noninterleaved planner cannot solve this problem?



Goal state:

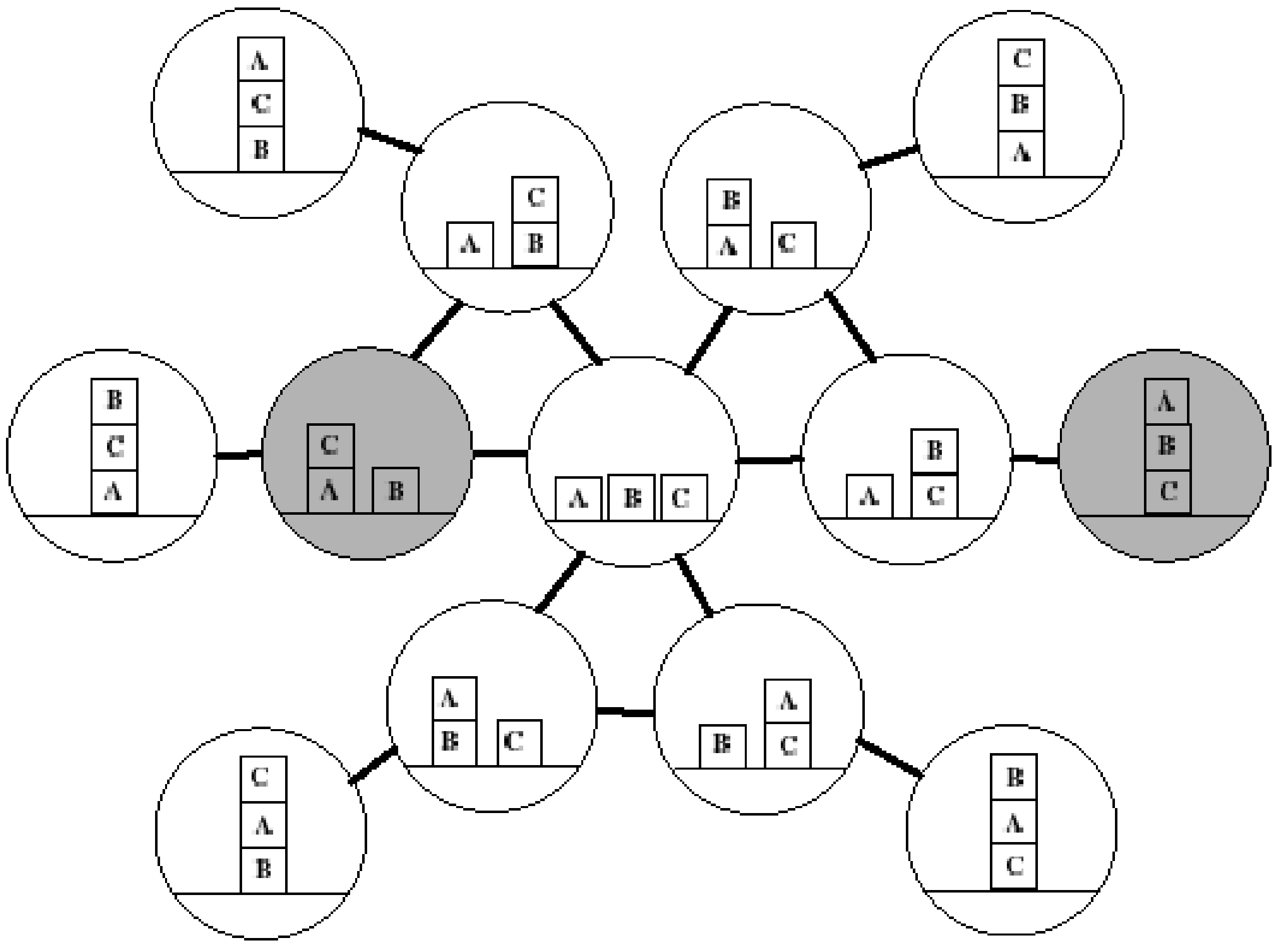
on(A,B)

on(B,C)

Outline

- STRIPS-like plan representation
- ➢ Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

Search space: Blocks world



Search the space of world states

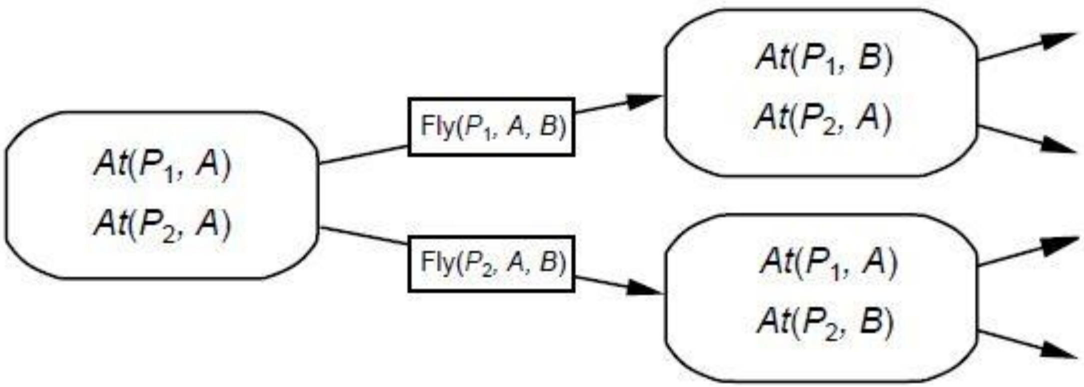
Planning as state space search

- Nodes: world states
- Arcs: actions
- Solution: path from the initial state to one state that satisfies the goal

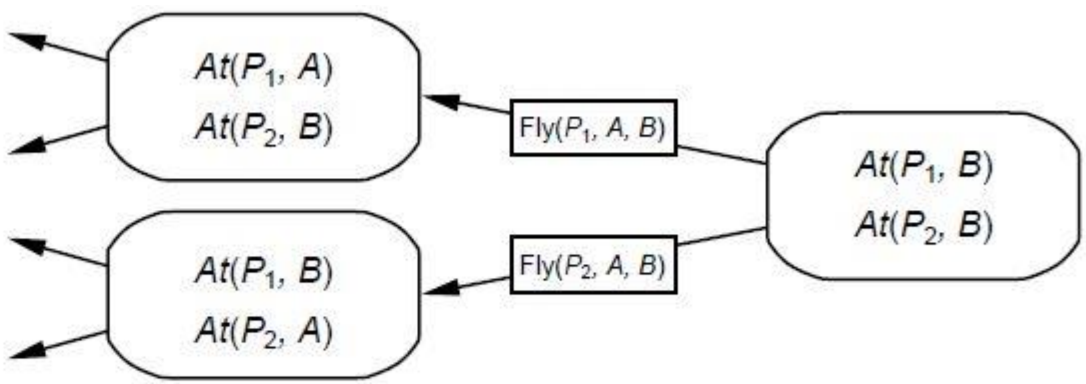
- Progression（前向规划）: forward search
- Regression（后向规划）: backward search

Planning algorithms 状态、空间太大，效率低下

- Progression: Forward state-space search



- Regression: Backward state-space search



规划算法衡量标准

Properties of planning algorithms

- Soundness 正确性
  - A planning algorithm is **sound** if all solutions are legal plans
    - All preconditions, goals, and any additional constraints are satisfied
- Completeness 完备
  - 可以找到解.
  - A planning algorithm is **complete** if a solution can be found
  - A planning algorithm is **strictly complete** if all solutions are included in the search space whenever one actually exists
- Optimality 最优解.
  - A planning algorithm is **optimal** if it maximizes a predefined measure of plan quality

Progression vs regression

- Both algorithms are
  - *sound* (they always return a valid plan)
  - *complete* (if a valid plan exists they will find one)
- Complexity $O(b^n)$ worst-case   b很大.(没有使用问题领域信息)
  - *where b = branching factor,*

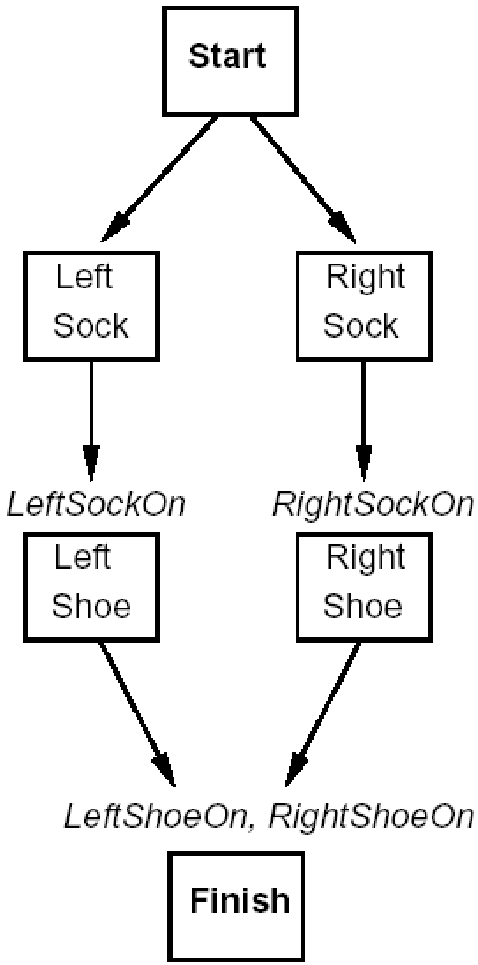
    n = number of “choose” operators
- Regression: often smaller b, focused by goals
- Progression: full state space to compute heuristics

Outline

- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

Total-order vs Partial-order plans

Partial Order Plan:



Total Order Plans:

Search the space of plans

Partial-Order Planning (POP) 搜索规划空间.

Generates partial-order plans

- Nodes are partial plans
- Links are plan refinements
- Solution is a node (not a path)

Follow the least commitment principle [Weld 94]

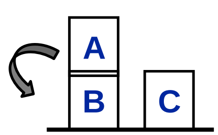
- Don’t commit to an order of actions until it is required

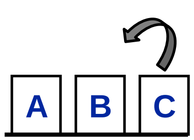
Partial plan representation

- Plan = (A, O, L), where
  - A: set of actions in the plan 动作
  - O: *temporal orderings* between actions (a < b) 必要的序
  - L: *causal links* linking actions via a literal 因果连接
- Causal Link: $Ap \xrightarrow{Q} Ac$ （因果连接）

  Action Ac (consumer) has precondition Q that is established in the plan by Ap (producer), e.g.

move-A-from-B-to-Table $\xrightarrow{\text{clear(B)}}$ *move-C-from-Table-to-B*





Threats to causal links and protection

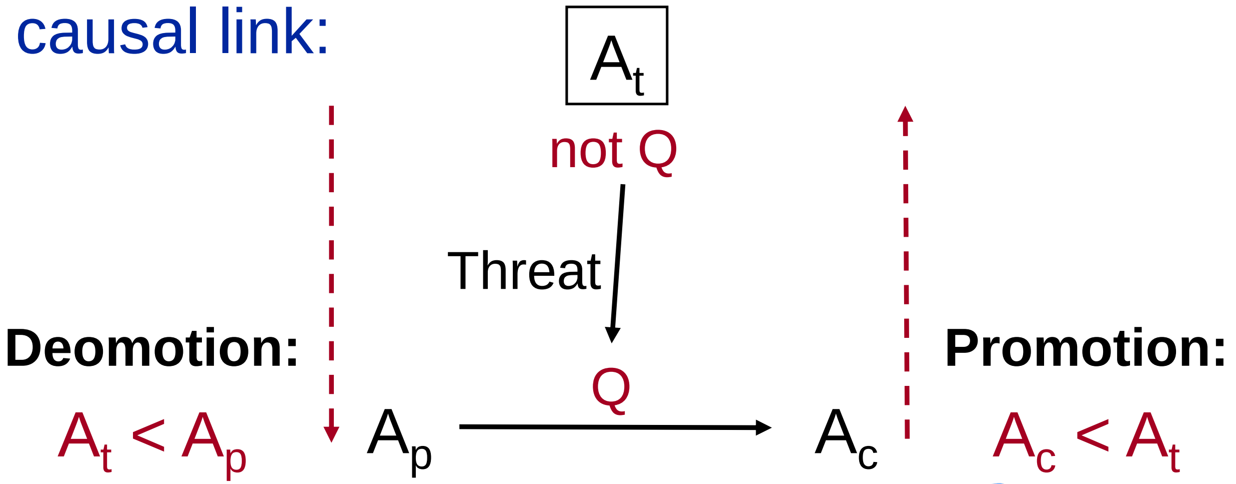
Step A_t threatens link (A_p, Q, A_c) if:

- A_t has (not Q) as an effect, and
- A_t could come between A_p and A_c, i.e.

  $O \cup (A_p < A_t < A_c)$ is consistent

  (Ordering **"<"** is not necessarily immediately before)

To protect causal link:



把At放在Ap之前或Ac之后，保持因果连接Q

Consistent plan in POP

A partial plan is consistent if:

无环

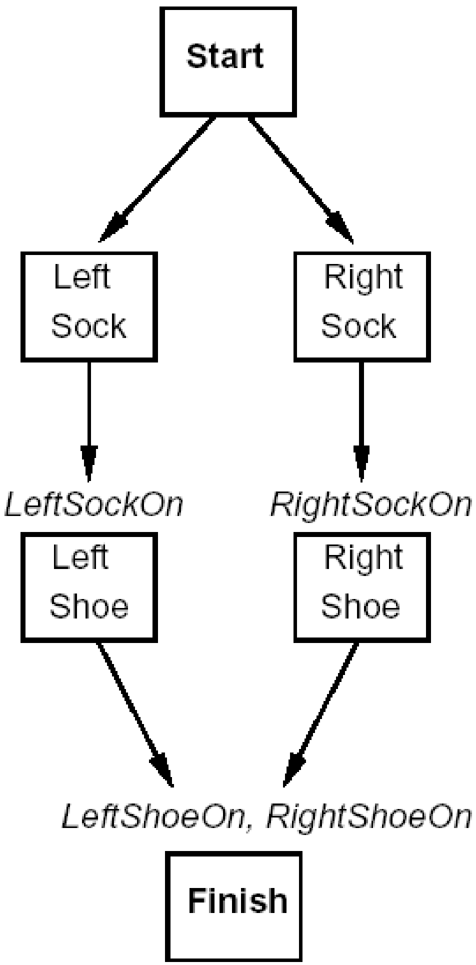
- There are no cycles in the ordering constraints and
- No conflicts with the causal links 无冲突

A consistent plan with no open preconditions is a **solution**:

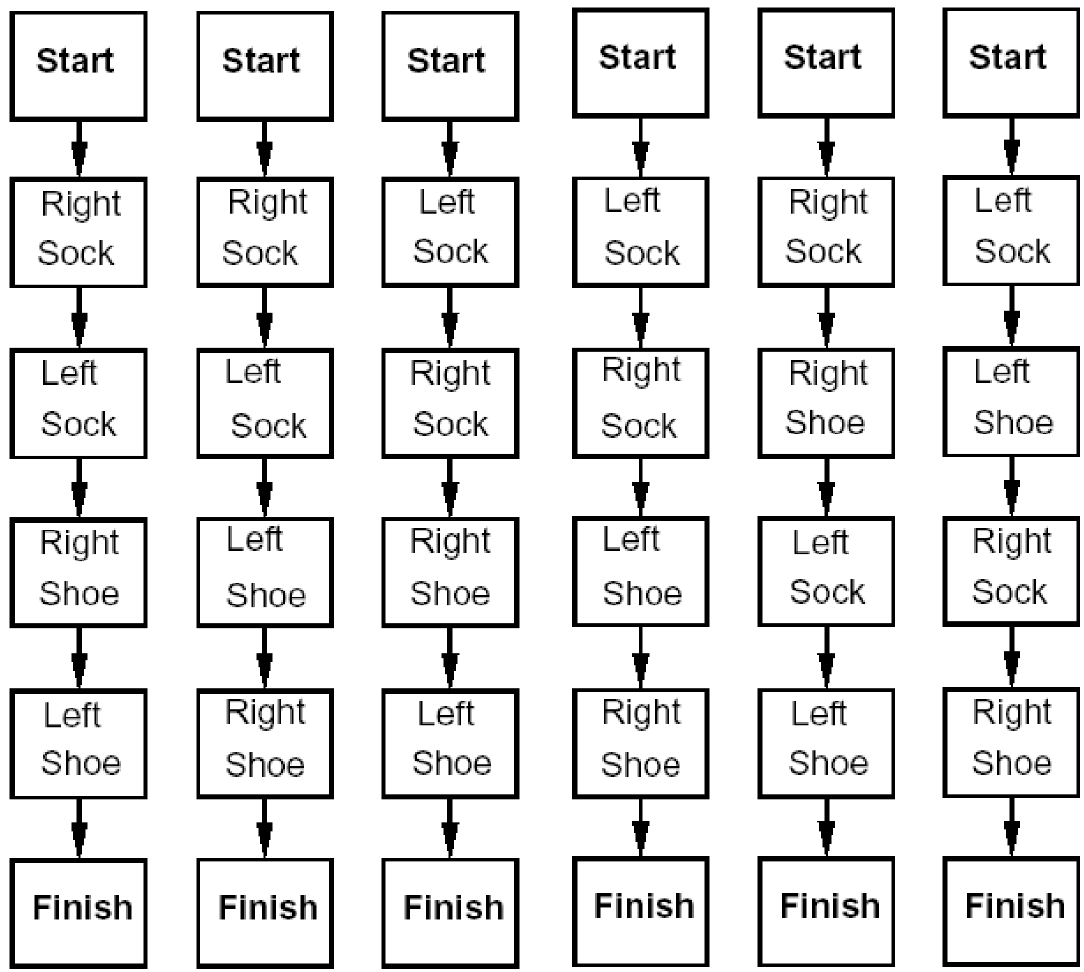
- Every linearization（线性化）of a partial-order solution is a total-order solution whose execution from the initial state will reach a goal state

Total-order vs Partial-order plans

Partial Order Plan:



Total Order Plans: （线性化）



Initial plan

For uniformity, represent initial state and goal with two special actions:

- Start (A_0):
  - no preconditions, 无前提条件.
  - initial states as effects,
  - must be the first step in the plan
- Finish (A_∞):
  - no effects 无结果
  - goals as preconditions
  - must be the last step in the plan

Agenda: set of open conditions （议程集）

- e.g. {(on(A,B), A_∞), (on(B,C), A_∞)}

POP algorithm

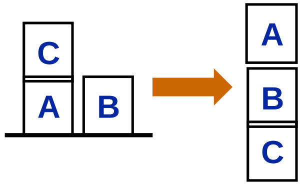
POP((A, O, L), agenda, actions)

Initial plan: {*Start*, *Finish*} and preconditions in *Finish* as open conditions

1. If **agenda** is empty, then **return** (A, O, L) 结束条件
2. Pick (Q, A_{need}) from **agenda** （子）目标
3. **Choose** an action A_{add} that adds effect Q 动作选择
   - If no such action exists, **fail**
   - Add the link $A_{add} \xrightarrow{Q} A_{need}$ to **L** and the ordering $A_{add}<A_{need}$ to **O**
   - If A_{add} is new, add it to **A** 规划扩充
4. Remove (Q, A_{need}) from **agenda**. If A_{add} is new, for each of its preconditions P add (P, A_{add}) to **agenda** 更新（子）目标
5. For every action A_t in A that threatens any causal link $A_p \longrightarrow A_c$ in **L**
   - **Choose** to add $A_t<A_p$ or $A_c<A_t$ to **O**
   - If neither choice is consistent, **fail**
6. POP((A, O, L), **agenda**, actions)

保护因果连接：
- 降级(Demotion): $\mathbf{A_t < A_p}$
- 升级(Promotion): $\mathbf{A_c < A_t}$

POP example: Sussman anomaly



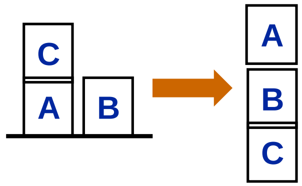
Start ($\mathbf{A_0}$)

on(C,A) on-table(A) on-table(B) clear(C) clear(B)

on-table(C) clear(A) on(A,B) on(B,C)

Finish ($\mathbf{A_\infty}$)

Work on open condition on(B,C)



Start ($\mathbf{A_0}$)

on(C,A) on-table(A) on-table(B) clear(C) clear(B)

clear(B) clear(C) on-table(B)

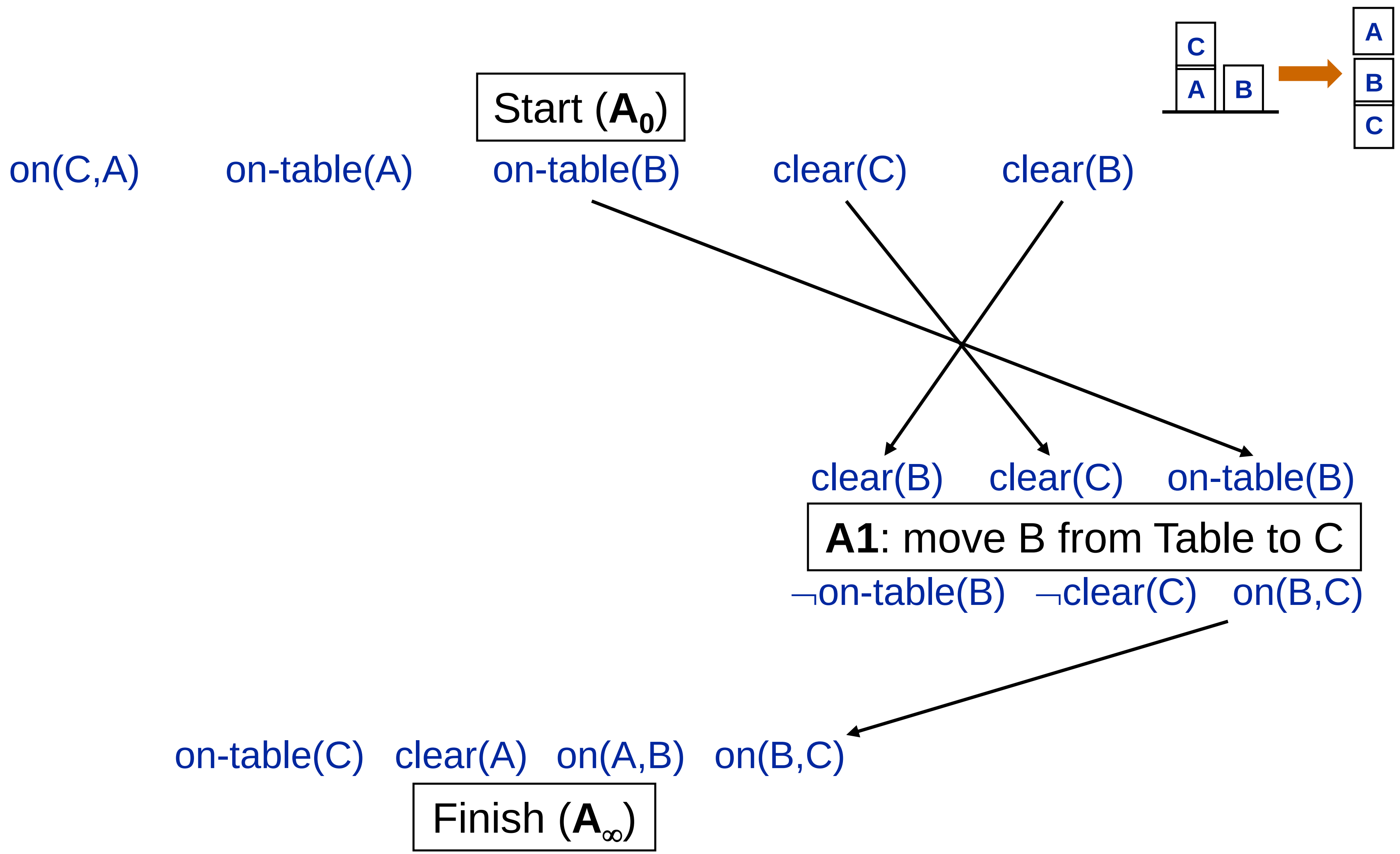
A1: move B from Table to C

¬on-table(B) ¬clear(C) on(B,C)

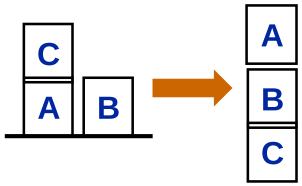
on-table(C) clear(A) on(A,B) on(B,C)

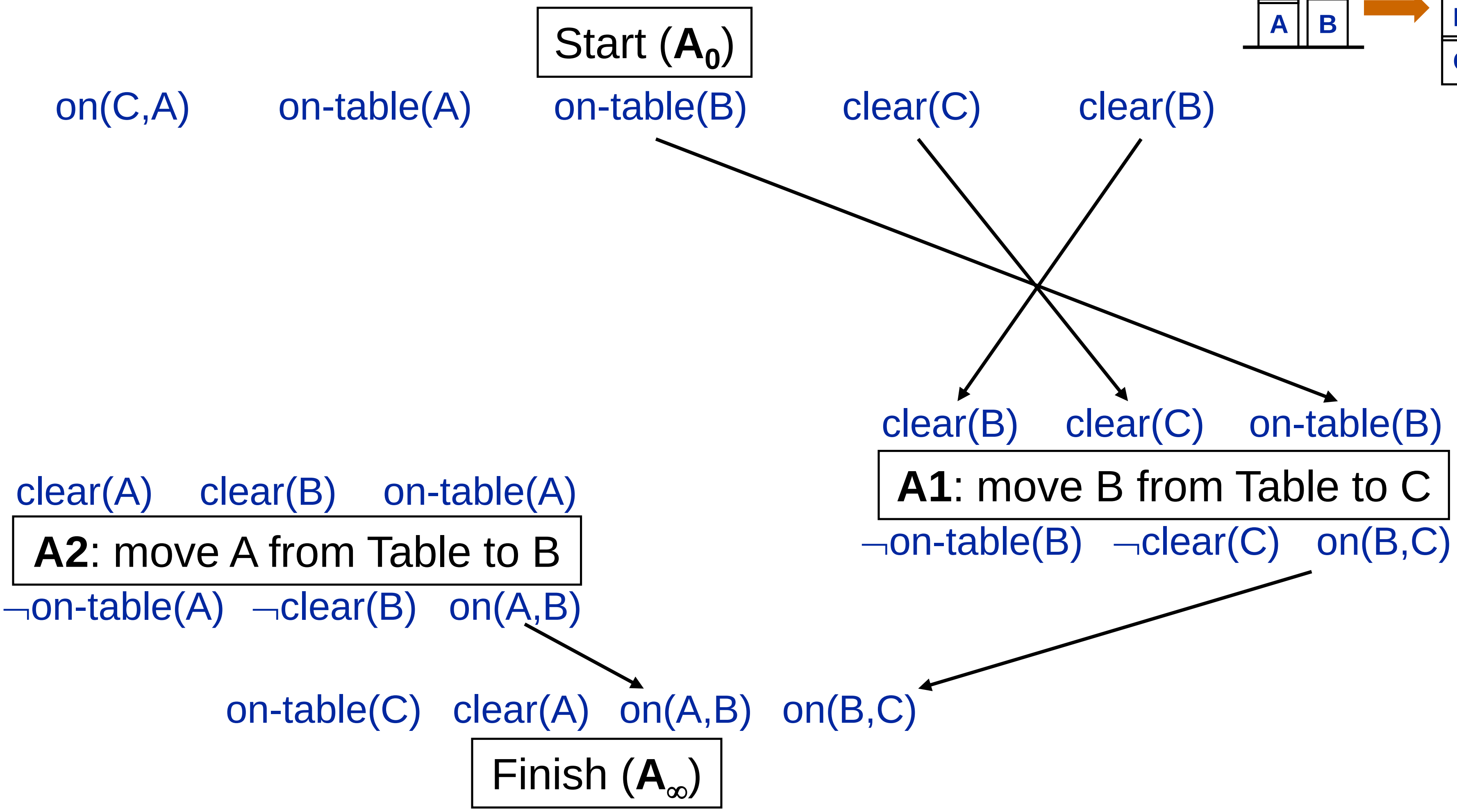
Finish ($\mathbf{A}_\infty$)

Work on open conditions clear(B), clear(C)...

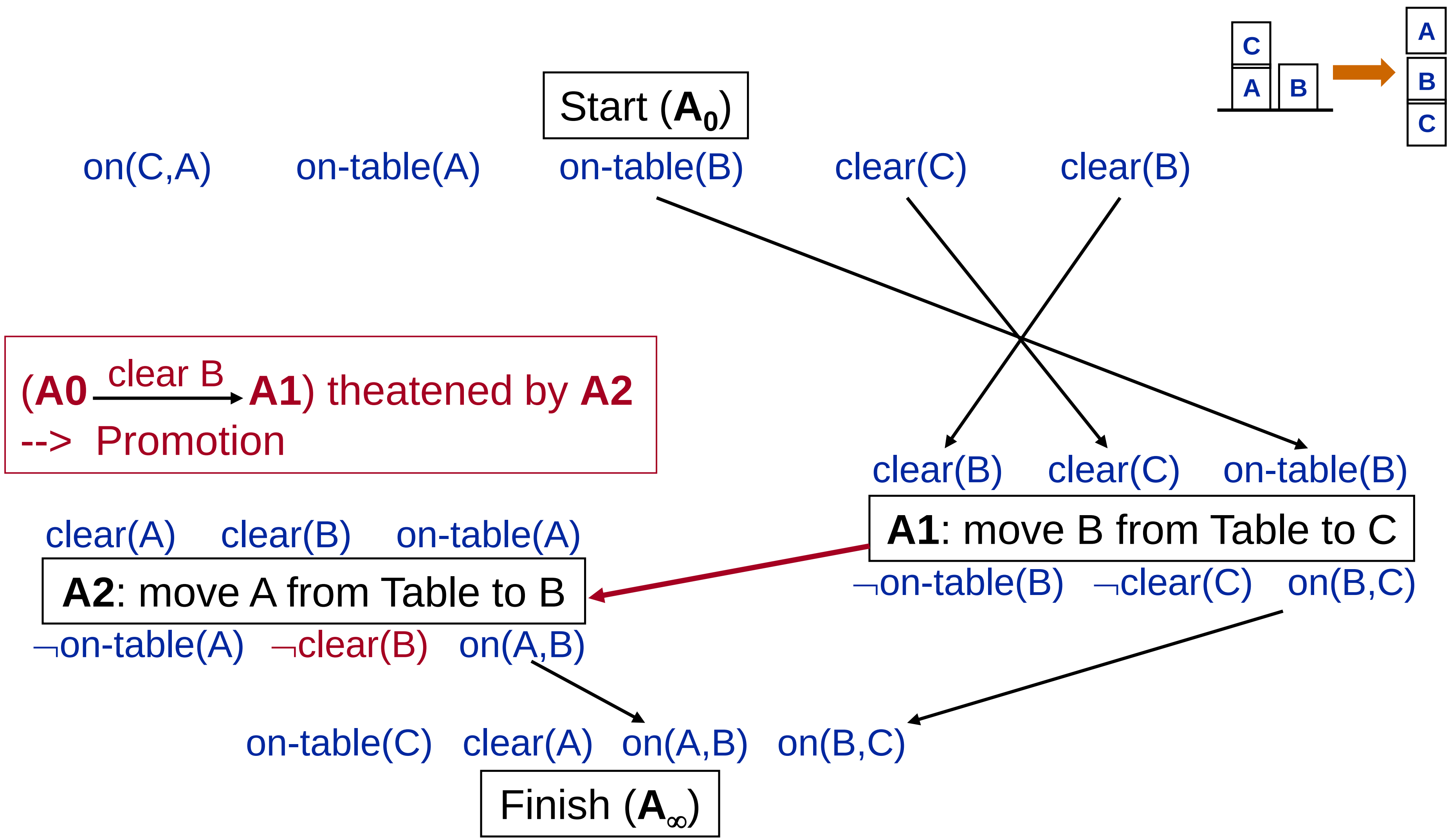


Work on open condition on(A,B)

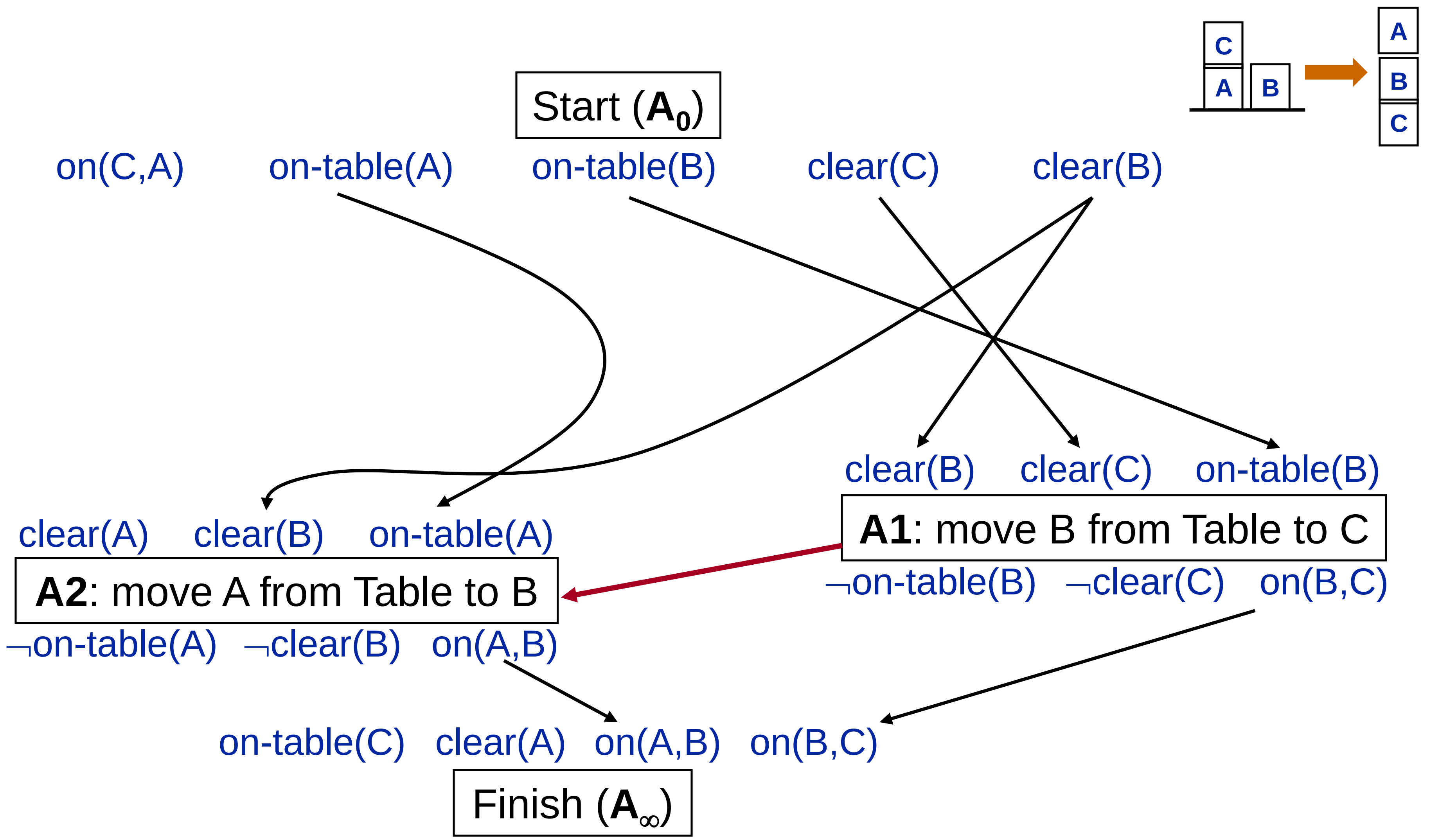




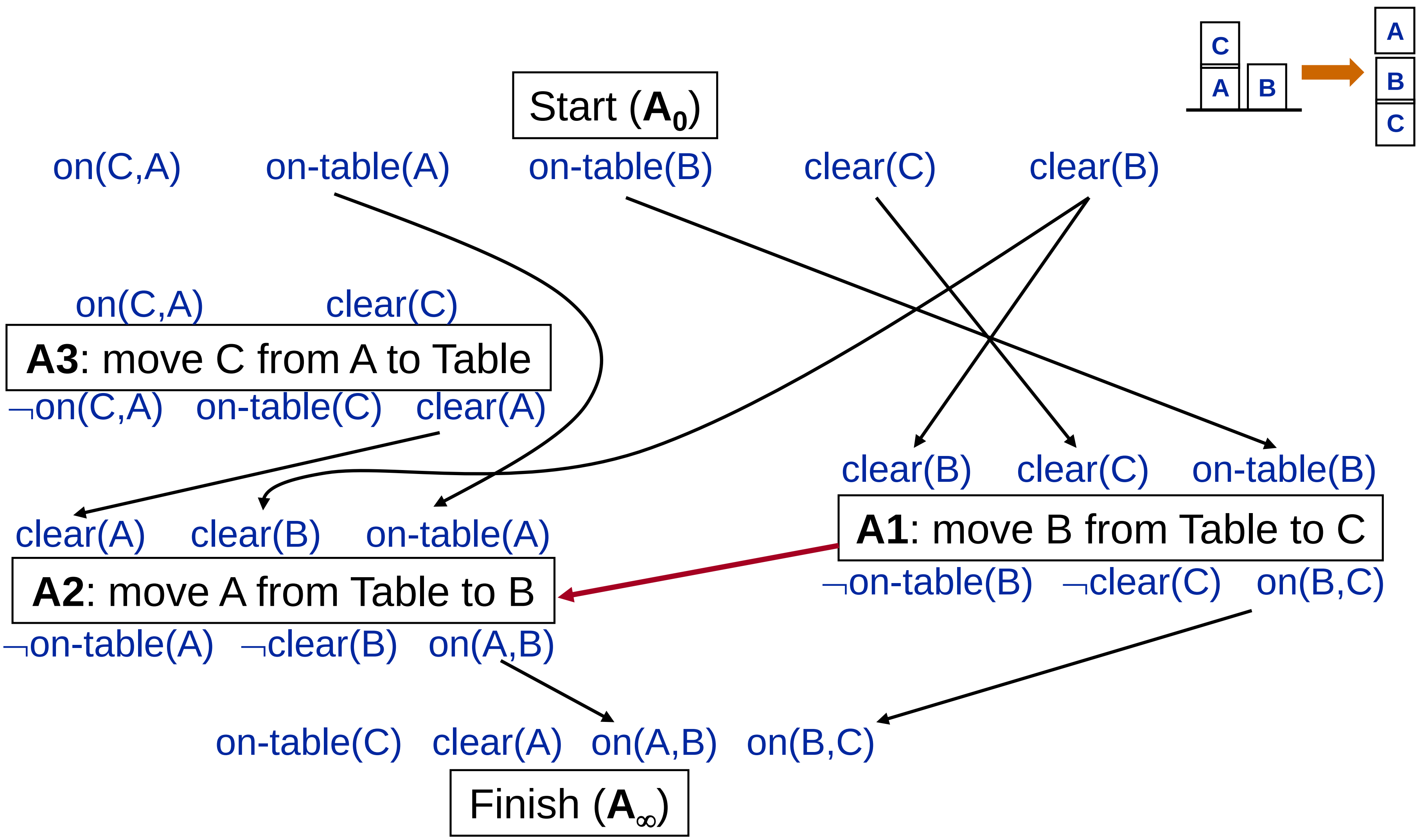
Protect causal link



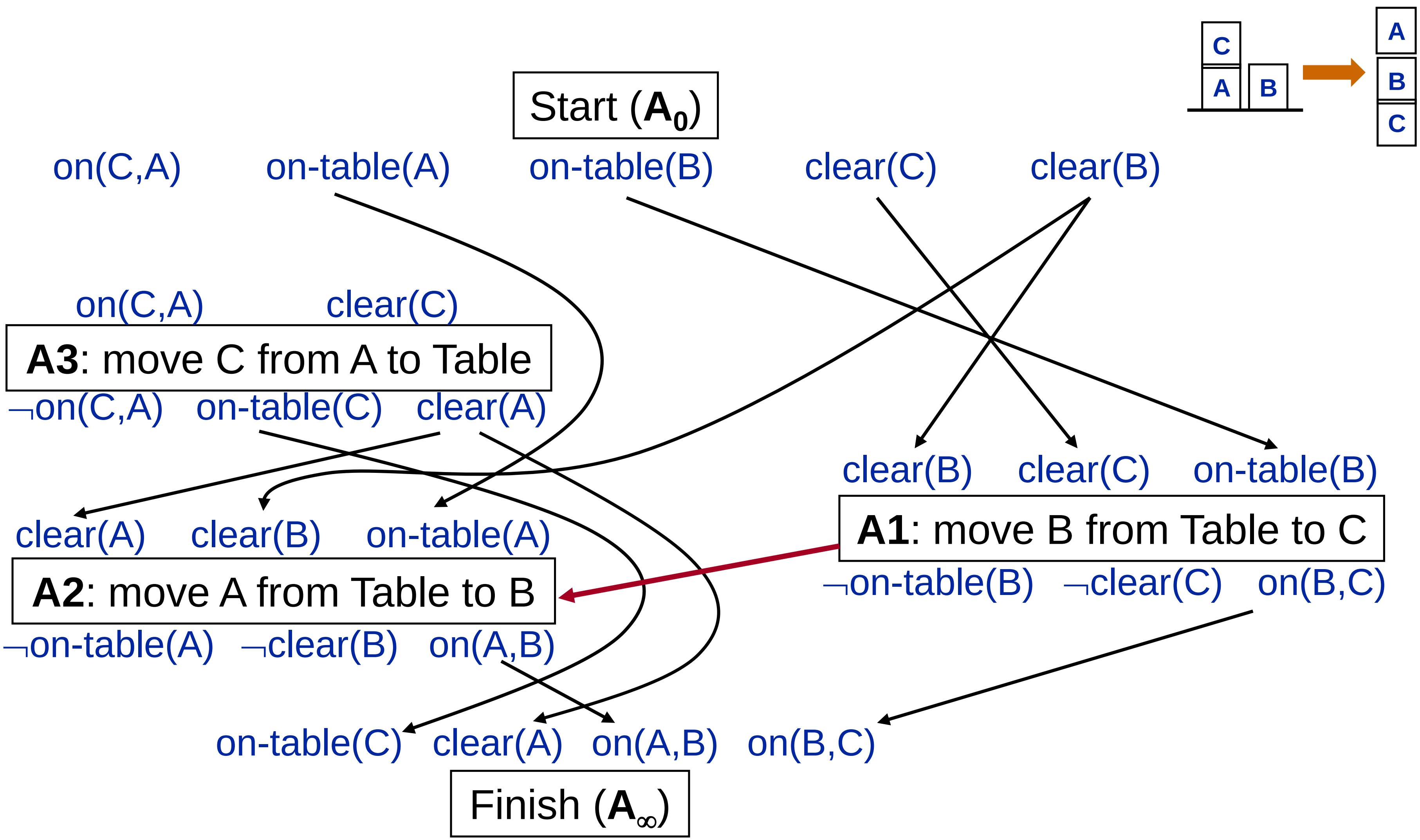
Work on conditions clear(B), on-table(A)



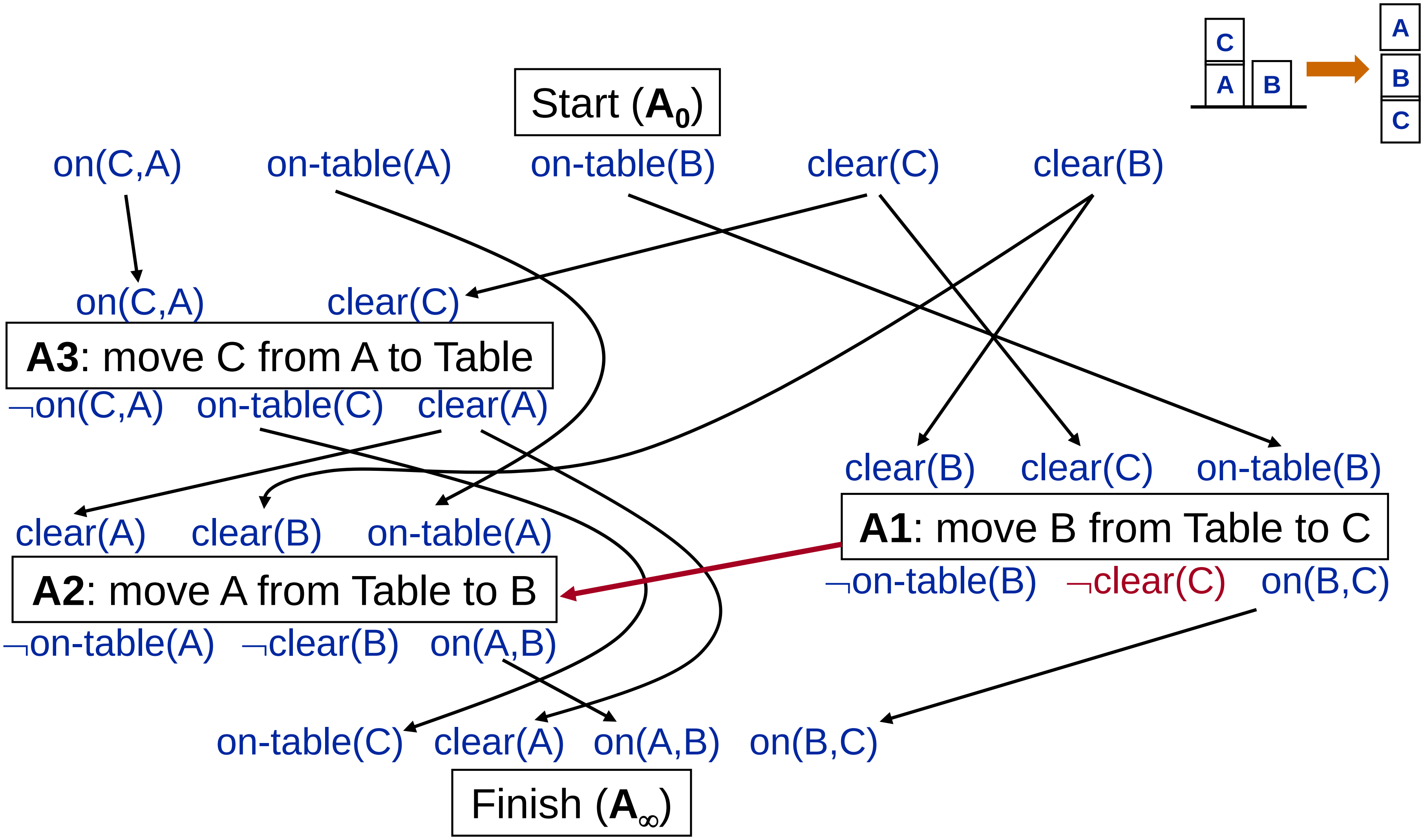
Work on condition clear(A)



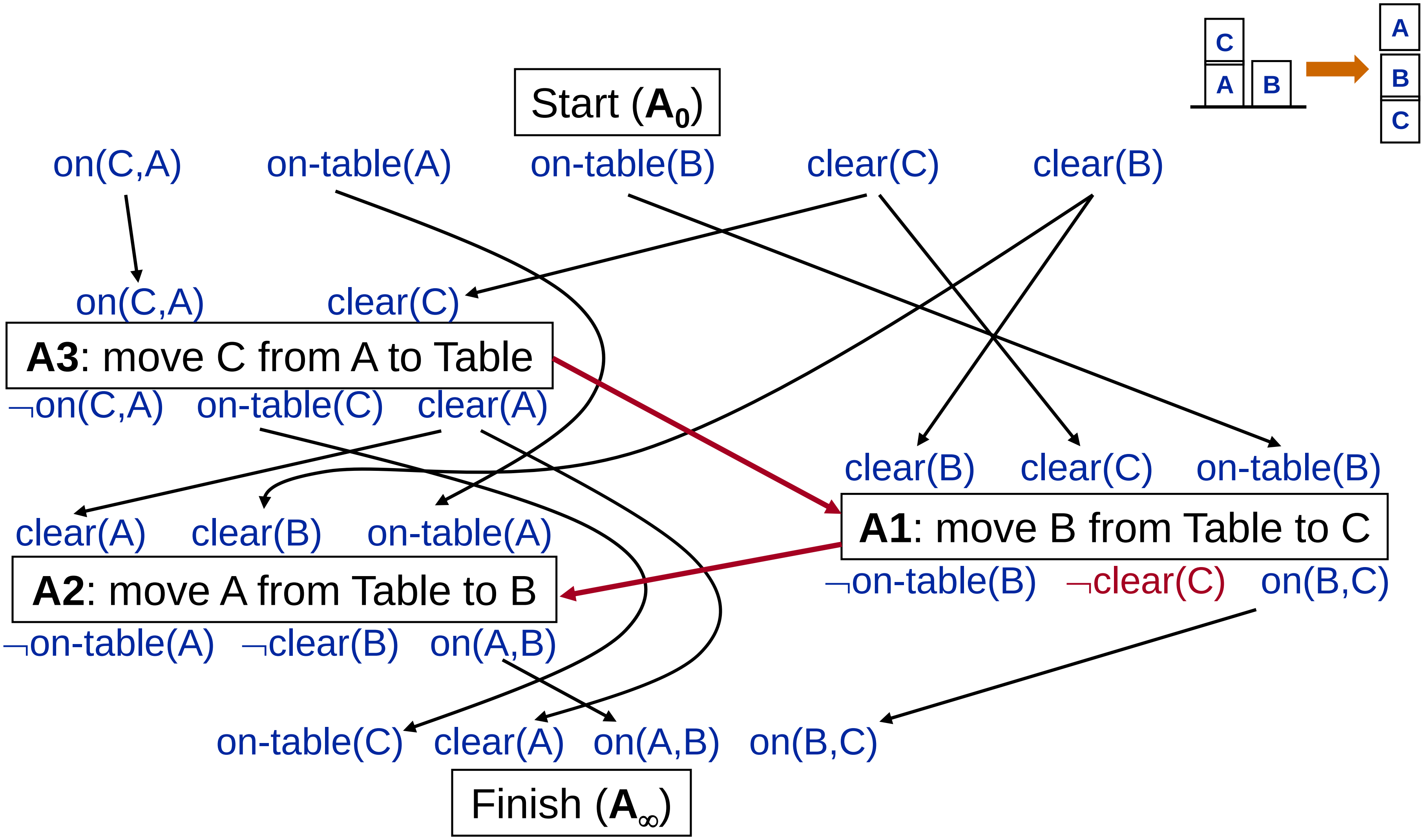
Work on conditions clear(A), on-table(C)



Work on conditions on(C,A), clear(C)



Protect causal link



Final plan step

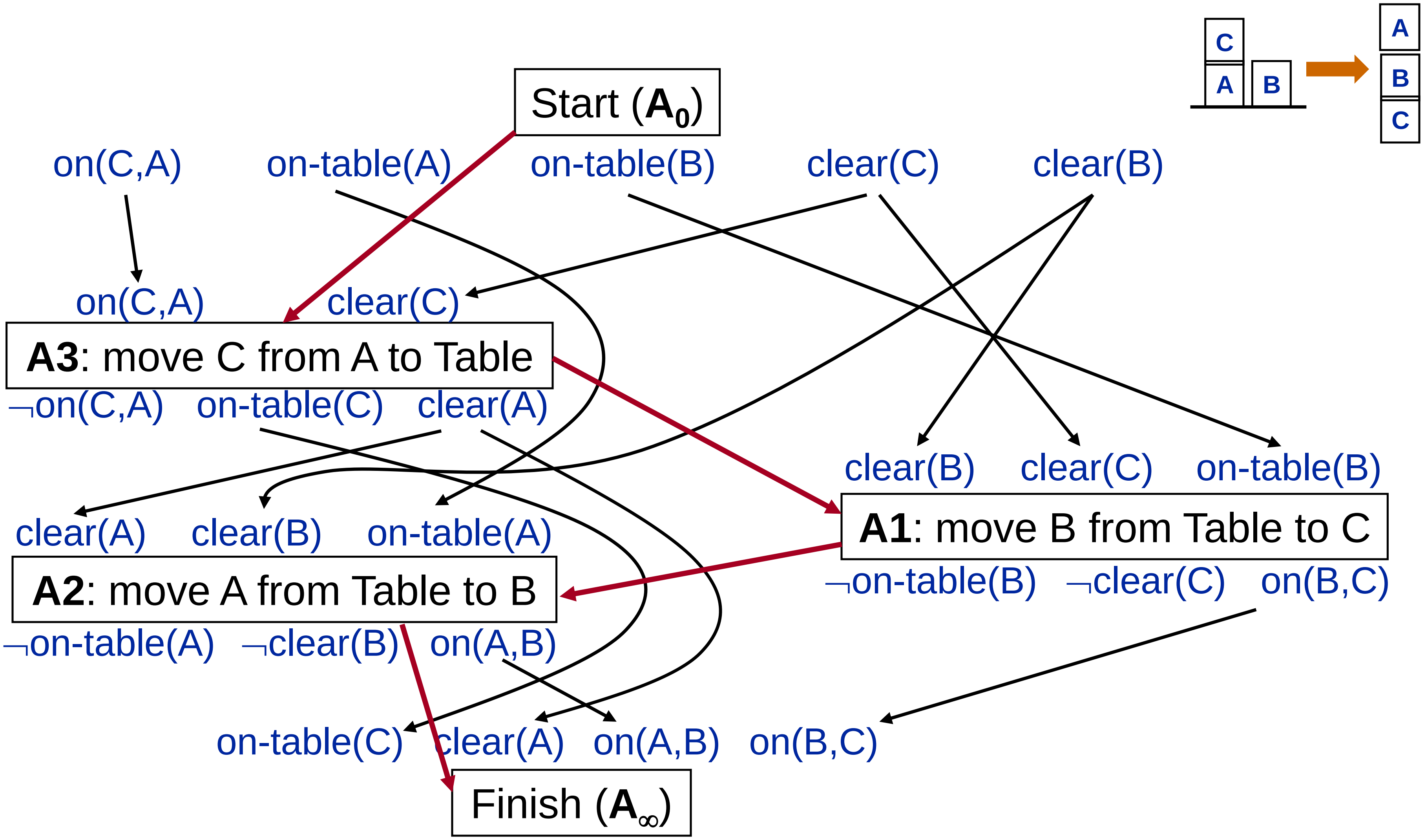
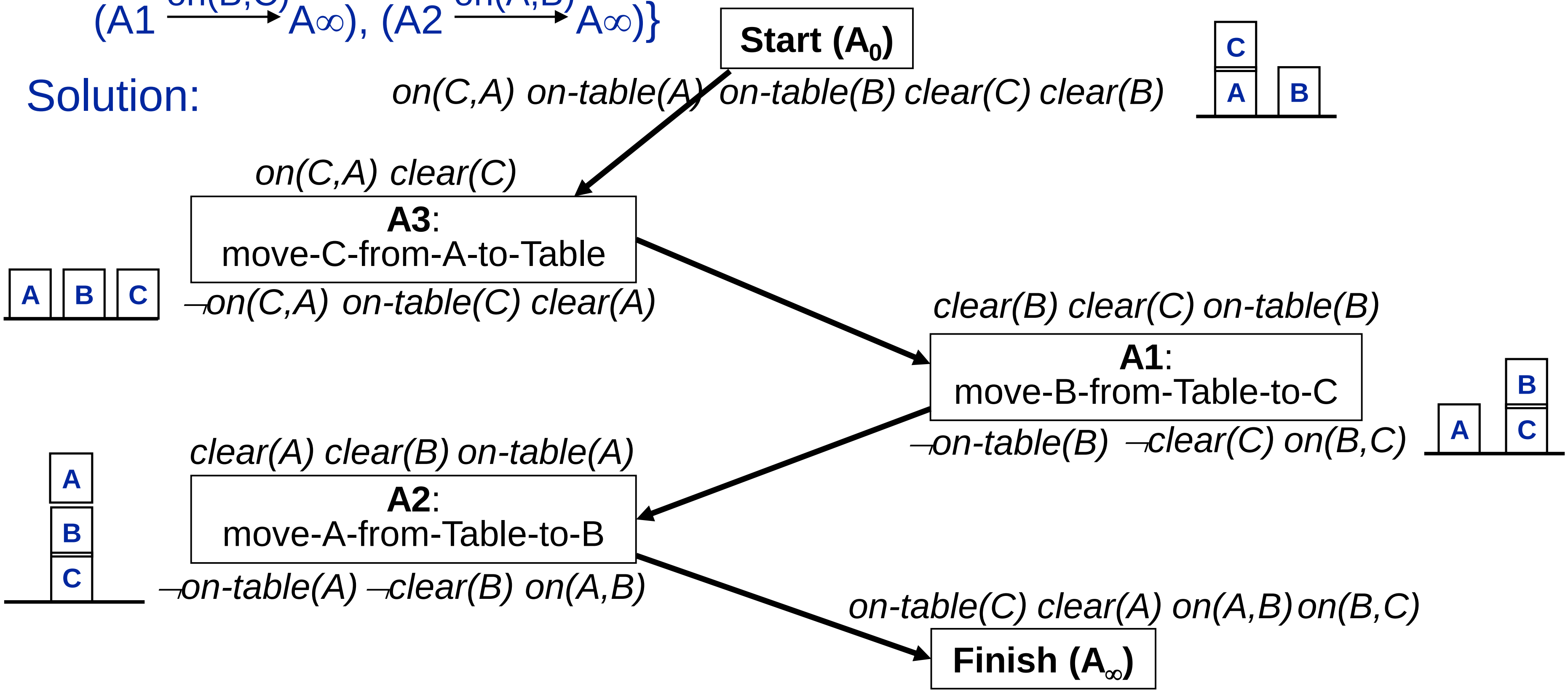


Illustration of solution plan

A: {A1, A2, A3, A0, A∞}; **O:** {A3<A1, A1<A2}

L: $\{(A0 \xrightarrow{on(C,A)} A3),\ (A0 \xrightarrow{clear(C)} A3),\ (A0 \xrightarrow{clear(B)} A1),\ (A0 \xrightarrow{clear(C)} A1),\ (A0 \xrightarrow{on\text{-}tbl(B)} A1),$
$(A0 \xrightarrow{clear(B)} A2),\ (A0 \xrightarrow{on\text{-}tbl(A)} A2),\ (A3 \xrightarrow{clear(A)} A2),\ (A3 \xrightarrow{on\text{-}tbl(C)} A\infty),\ (A3 \xrightarrow{clear(A)} A\infty),$
$(A1 \xrightarrow{on(B,C)} A\infty),\ (A2 \xrightarrow{on(A,B)} A\infty)\}$

Solution:



POP exercise

- For this problem of putting one's shoes and socks, apply POP to this problem. Show the final plan, and corresponding A, O and L for this problem.

  (the ordering constraints that put every other action after *Start* and before *Finish* can be omitted in O)

- The planning problem is described as follows:

  Initial state: *Empty*

  Goal: *RightShoeOn*, *LeftShoeOn*

  Action *RightShoe*

  Precondition: *RightSockOn*

  Effect: *RightShoeOn*

  Action *RightSock*

  Effect: *RightSockOn*

  Action *LeftShoe*

  Precondition: *LeftSockOn*

  Effect: *LeftShoeOn*

  Action *LeftSock*

  Effect: *LeftSockOn*

Properties of POP

POP is *sound* and *complete*

- Can be much faster than state-space planning, because of no need to backtrack over goal orderings (so less branching is required)
- Although it is more expensive per node and makes more choices than Regression, reduction in branching size often gains more
  - Larger *n* but smaller *b*

Flexibility gained by partial order

- Can be very useful to agent when the world fails to cooperate
- Make it easier to combine smaller plans into larger ones
  - Each plan can reorder its actions to avoid conflict with other plans

Partial-order (POP) vs State-space planning

Complexity: $O(b^n)$ worst-case

- Non-deterministic choices (n):
  - Progression, Regression: n = |actions|
  - POP: n = |preconditions| + |link protection| n 更大
  - Generally an action has several preconditions
- Branching factor (b)

  POP has smaller b:

  - No backtrack due to goal ordering
  - Least commitment: no premature step ordering

Comparison

	State Space	Plan Space
Algorithm	Progression Regression	POP
Nodes	World States	Partial Plans
Edges/ Transitions	Actions E.g. in blocks world: ▪ move-A-from-B-to-C ▪ move-B-from-A-to-Table ▪ move-C-from-B-to-A ▪ …	Plan refinements: ▪ Step addition ▪ Step reuse ▪ Demotion ▪ Promotion

More expressive action representation

UCPOP [Penberthy and Weld 92]

- Actions with variables

  Actions: Move-C-from-A-to-Table
  Move-A-from-B-to-C
  → Move ?b from ?x to ?y

- Conditional effects（条件结果）

  Effects: (and (on ?b ?y) (not (on ?b ?x)) (clear ?x)
  (when (= ?y Table) (clear ?y)))

- Disjunctive（析取） preconditions

  Preconditions: (and (on ?b ?x)
  (or (clear ?y) (big-and-flat ?y)))

- Universal quantification（全称量词）

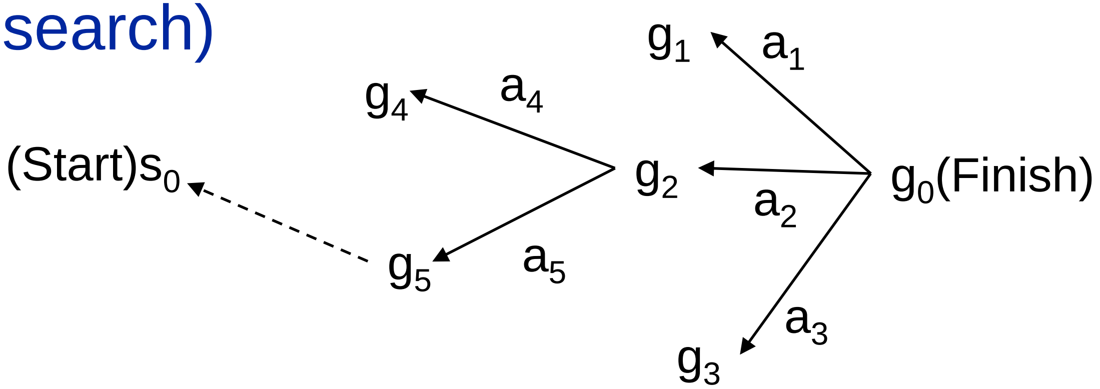
Outline

- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

History and motivation

- Before GraphPlan, mostly work on PSP-like planners:
  - POP, SNLP, UCPOP, etc
- Because GraphPlan was so much faster, many sub-sequent planning algorithms used ideas from it
  - IPP, STAN, GraphHTN, SGP, Blackbox, Medic, TGP, LPG
  - Many of them are much faster than the original Graphplan

Regression (backward search) may try lots of actions that can't be reached from the initial state



Main ideas

- A big source of inefficiency in search algorithms is the *branching factor*
  - the number of children of each node

- GraphPlan reduces the branching factor:

1. Create a *relaxed problem*
   - Remove some restrictions of the original problem
     - Want the relaxed problem to be easy to solve (polynomial time)
   - The solutions to the relaxed problem will include all solutions to the original problem
2. Do a modified search for the original problem
   - Restrict its search space to include only those actions that occur in solutions to the relaxed problem

GraphPlan

GraphPlan [Blum and Furst 97]

- Preprocessing before engaging in search
- Construct a *planning graph* to record constraints on possible plans
- Forward search combined with backward search, incl. two stages:
  - Extend: extend the planning graph at each time step
  - Search: Use the planning graph to constrain search for a solution plan
- Graphplan either finds a valid plan or concludes there is no solution exists

GraphPlan algorithm

Procedure GraphPlan:

- For k = 0, 1, 2, …

<Graph expansion>: 规划图扩展

- Construct a planning graph that contains k levels
- Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence
- If it does, then

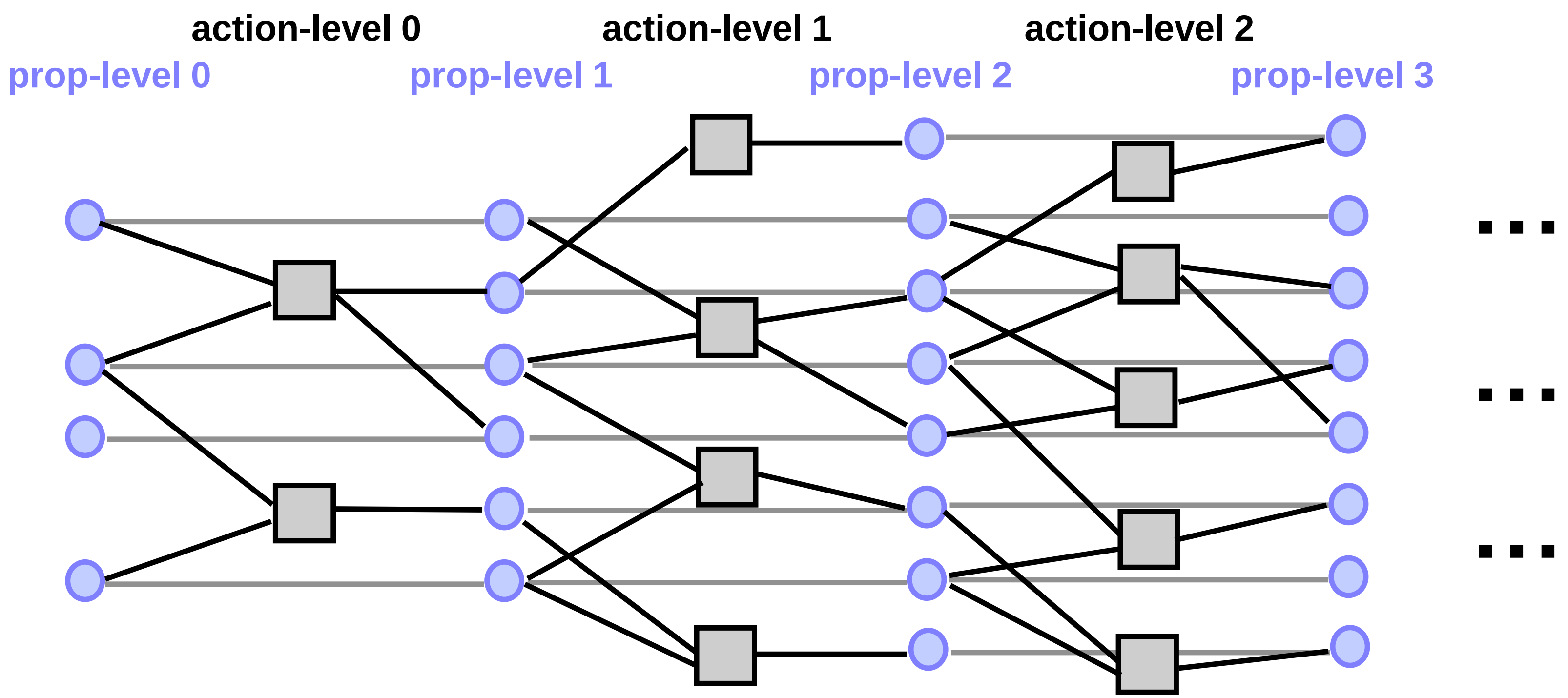
relaxed problem

<Solution extraction>: 规划解提取

- Backward search, modified to consider only the actions in the planning graph
- If a solution is found, then return *solution* 结束条件

A planning graph（规划图）

- Consist of a sequence of **levels** that correspond to time steps in the plan, where level 0 is the initial state
- Each level contains a set of propositional literals and a set of actions, with edges connecting action preconditions and effects



Expanding a planning graph - Actions

- To expand an action-level *i*:
  - Add each instantiated action, for which all of its preconditions are present at proposition-level *i* AND

    no two of its preconditions are exclusive
  - Add all the no-op actions 空操作
- Determine the mutual exclusive (mutex, 互斥) actions

Expanding a planning graph – Propositions

- To expand a proposition-level $i+1$:
  - Add all the effects of the inserted actions at action-level i: distinguishing *add and delete effects*
- Determine the mutual exclusive (mutex, 互斥) propositions

Determining mutex relations（互斥关系）

Two actions A and B are *mutex* at an action-level, if:

前提和结果冲突

- **Interference**: A (or B) disables a *precondition* or an *effect* of B (or A)

前提条件矛盾.

- **Competing needs**: A and B have *mutex preconditions*

Two propositions *p* and *q* are *mutex* at a proposition-level if

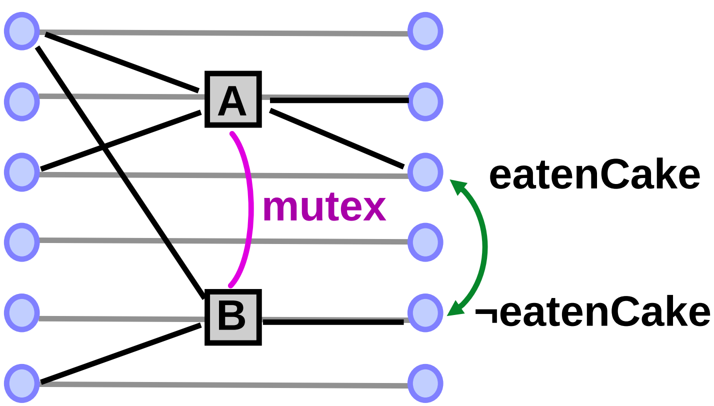
- **Negation**: *p* (or *q*) is the *negation* of *q* (or *p*) 互为否定
- **Inconsistent support**: *all ways* of achieving them are *mutex* (i.e. all the actions that add *p* are mutual exclusion of all the actions that add *q*)

Example: mutex actions

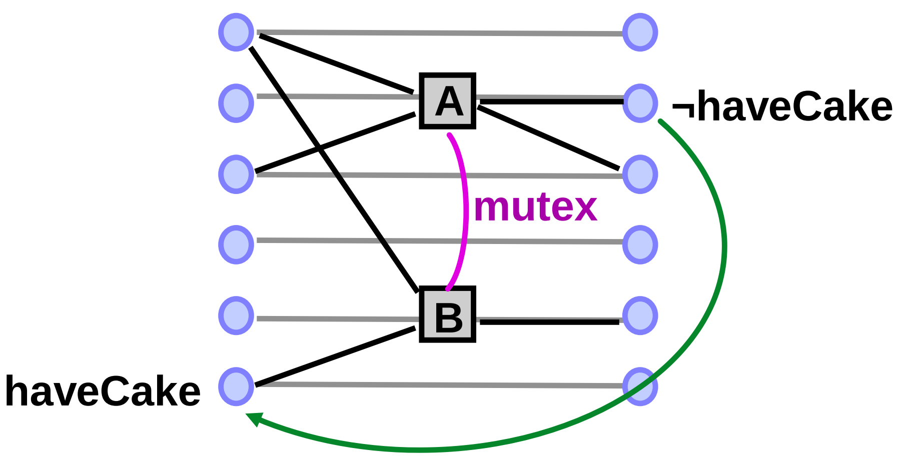
Two actions A and B are *mutex* at an action-level, if:

- **Interference**: A (or B) disables a *precondition* or an *effect* of B (or A)
- **Competing needs**: A and B have *mutex preconditions*

Interference (inconsistent effects):



Interference (precondition-effect):

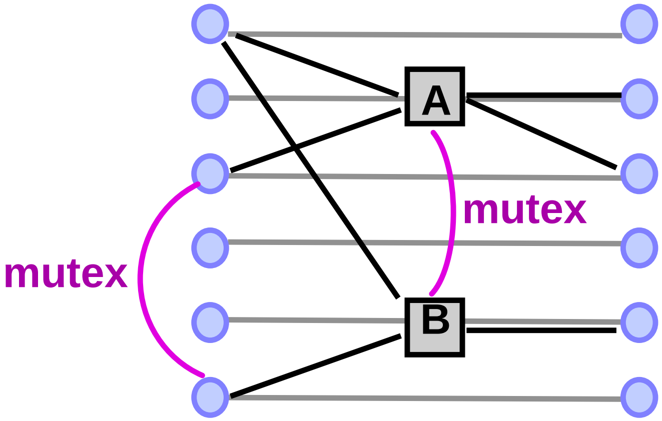


Example: mutex actions

Two actions A and B are *mutex* at an action-level, if:

- **Interference**: A (or B) disables a *precondition* or an *effect* of B (or A)
- **Competing needs**: A and B have *mutex preconditions*

Competing needs (mutex preconditions):

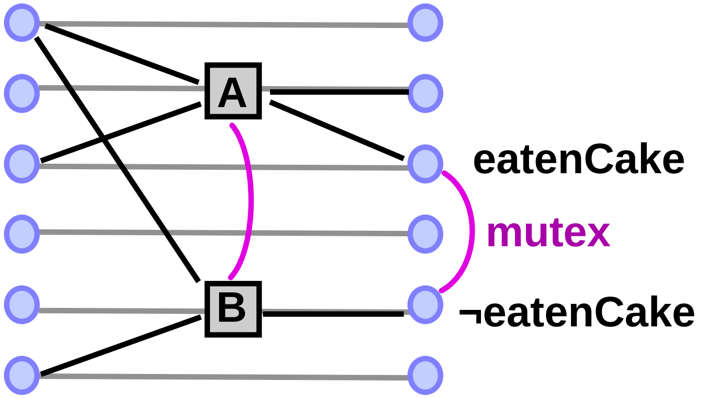


Example: mutex propositions

Two propositions *p* and *q* are *mutex* at a proposition-level if

- **Negation**: *p* (or *q*) is the *negation* of *q* (or *p*)
- **Inconsistent support**: *all ways* of achieving them are *mutex* (i.e. all the actions that add *p* are mutual exclusion of all the actions that add *q*)

Negation:

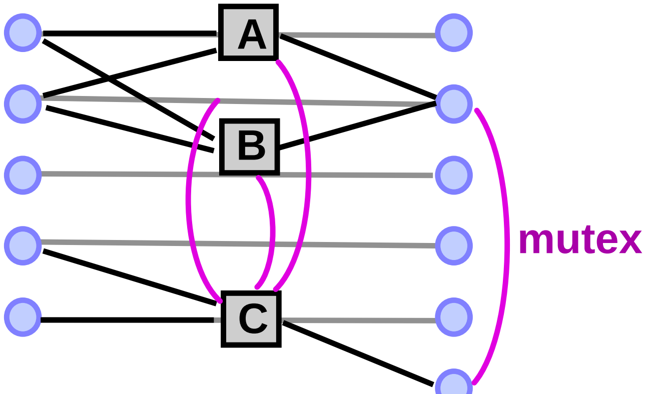


Example: mutex propositions

Two propositions *p* and *q* are *mutex* at a proposition-level if

- **Negation**: *p* (or *q*) is the *negation* of *q* (or *p*)
- **Inconsistent support**: *all ways* of achieving them are *mutex* (i.e. all the actions that add *p* are mutual exclusion of all the actions that add *q*)

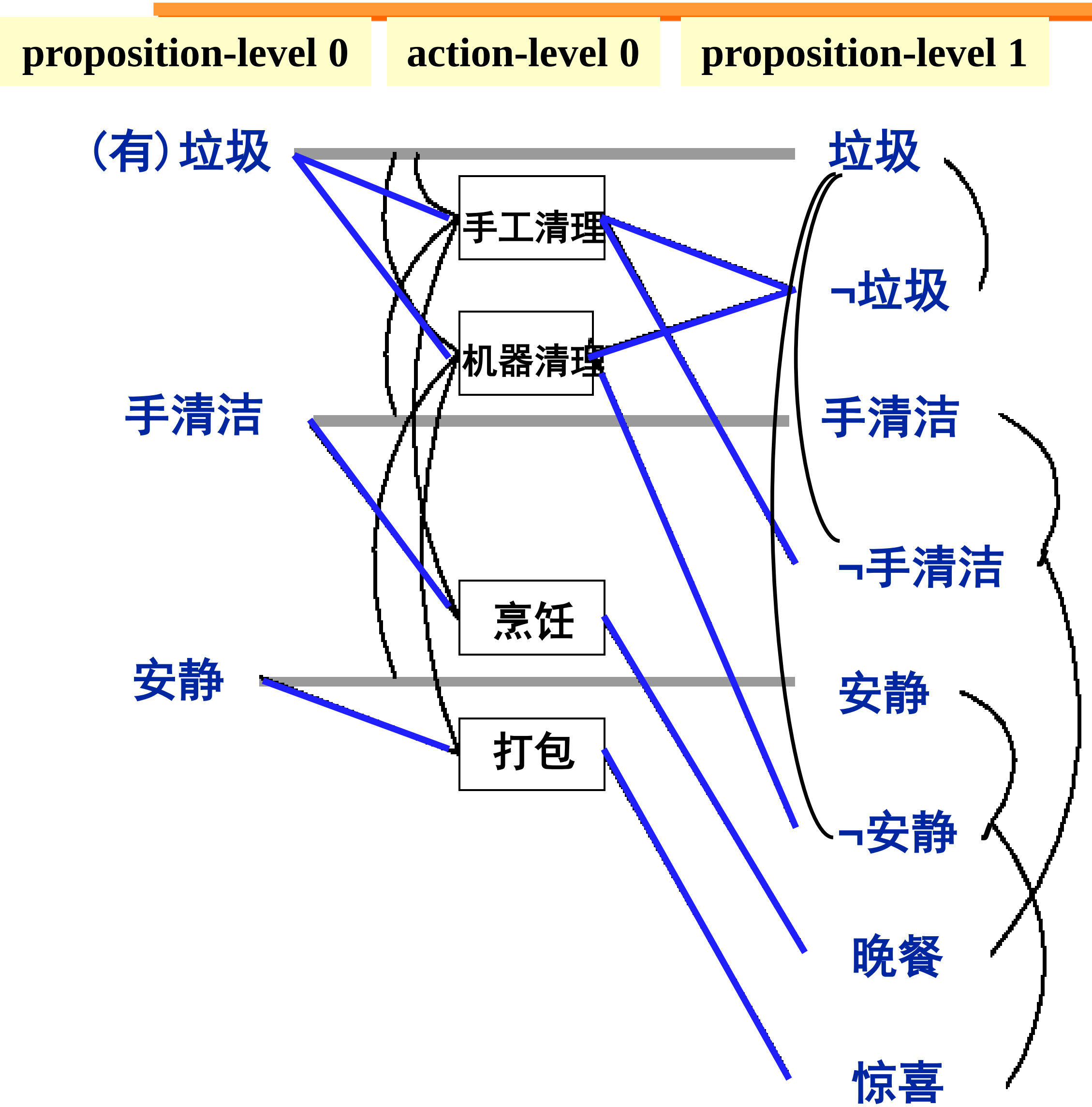
Inconsistent support:



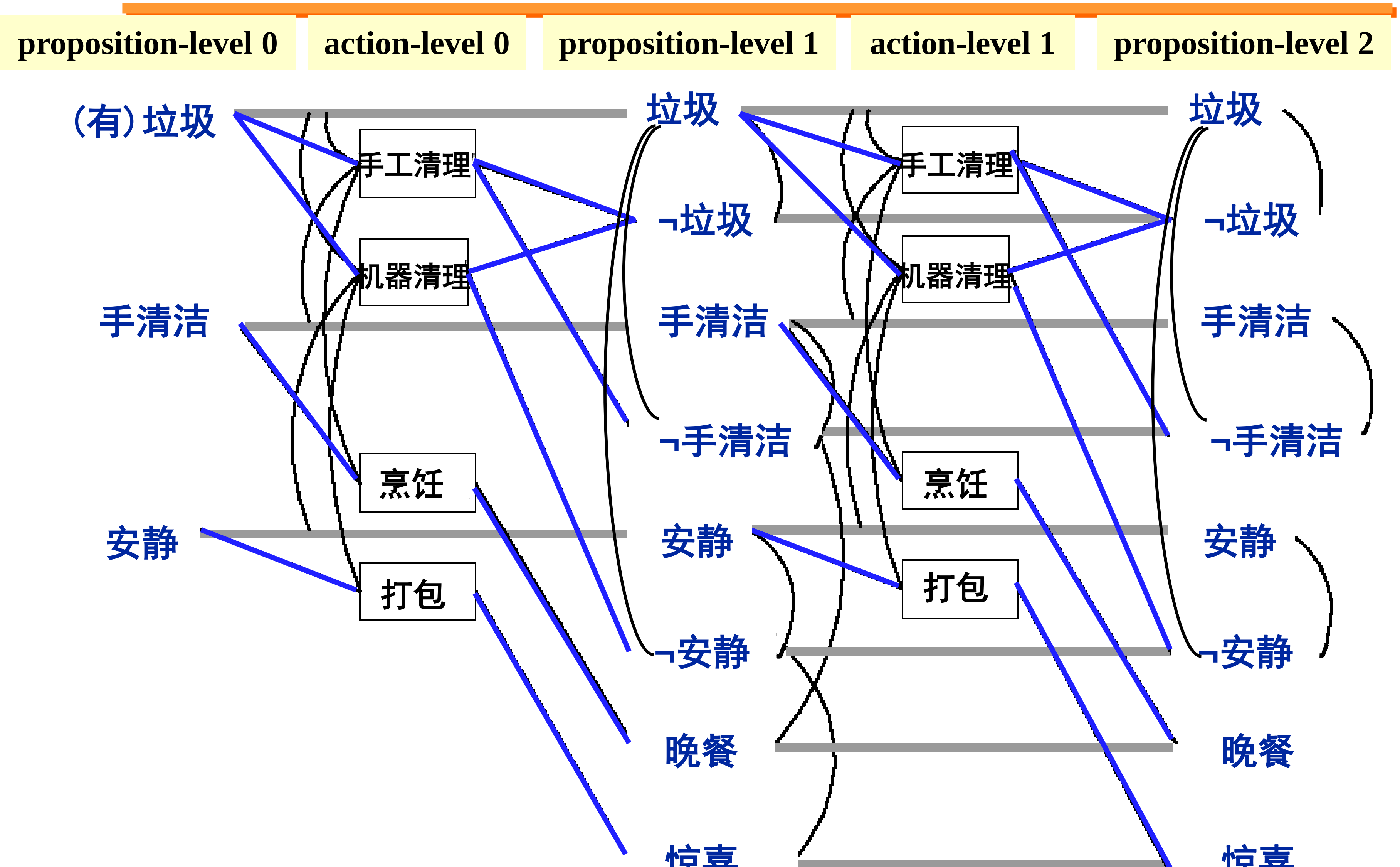
Dinner date example（为恋人准备惊喜晚餐）

- Initial state: (有) 垃圾, 手清洁, 安静
- Goal: 晚餐, 惊喜, ¬垃圾
- Actions:
  - 烹饪: precondition (手清洁)
    effect (晚餐)
  - 打包: precondition (安静)
    effect (惊喜)
  - 手工清理: precondition (垃圾)
    effect (¬垃圾, ¬手清洁)
  - 机器清理: precondition (垃圾)
    effect ((¬垃圾, ¬安静)

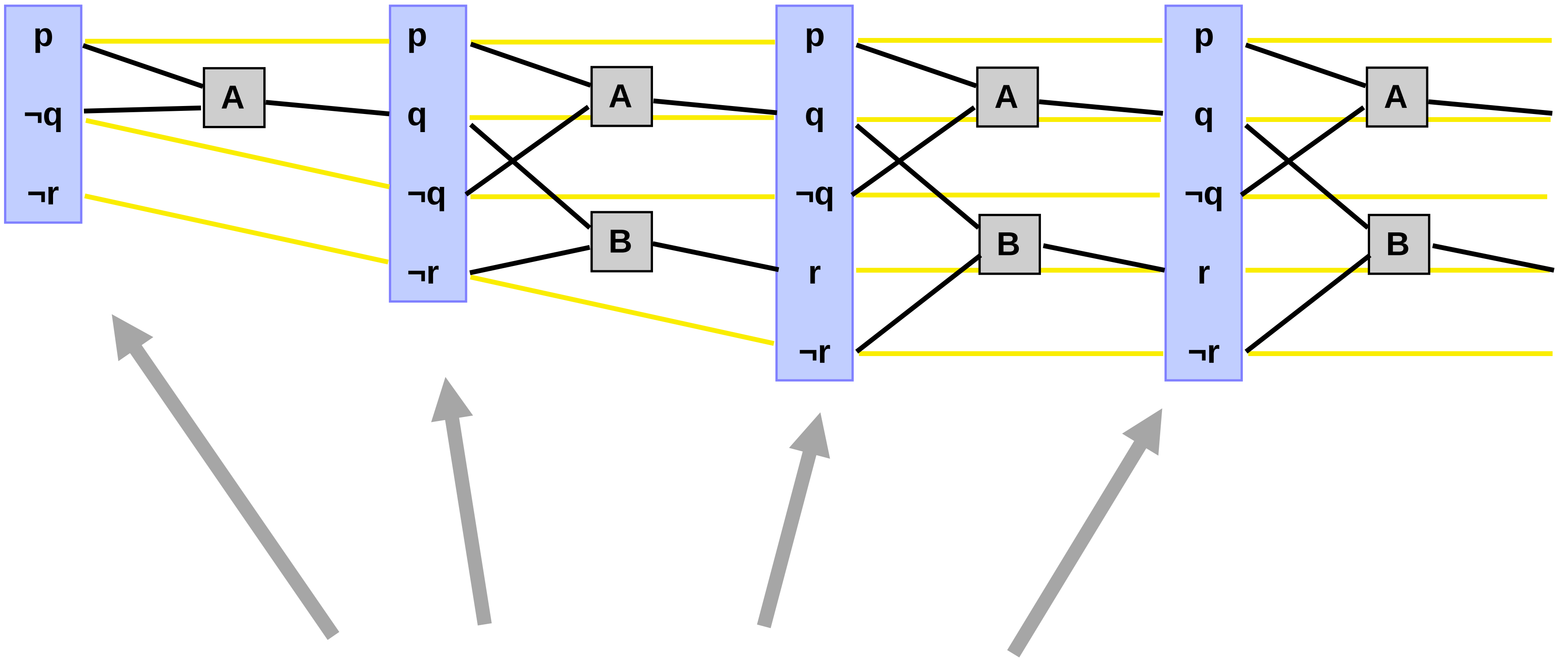
Dinner date example



Dinner date example



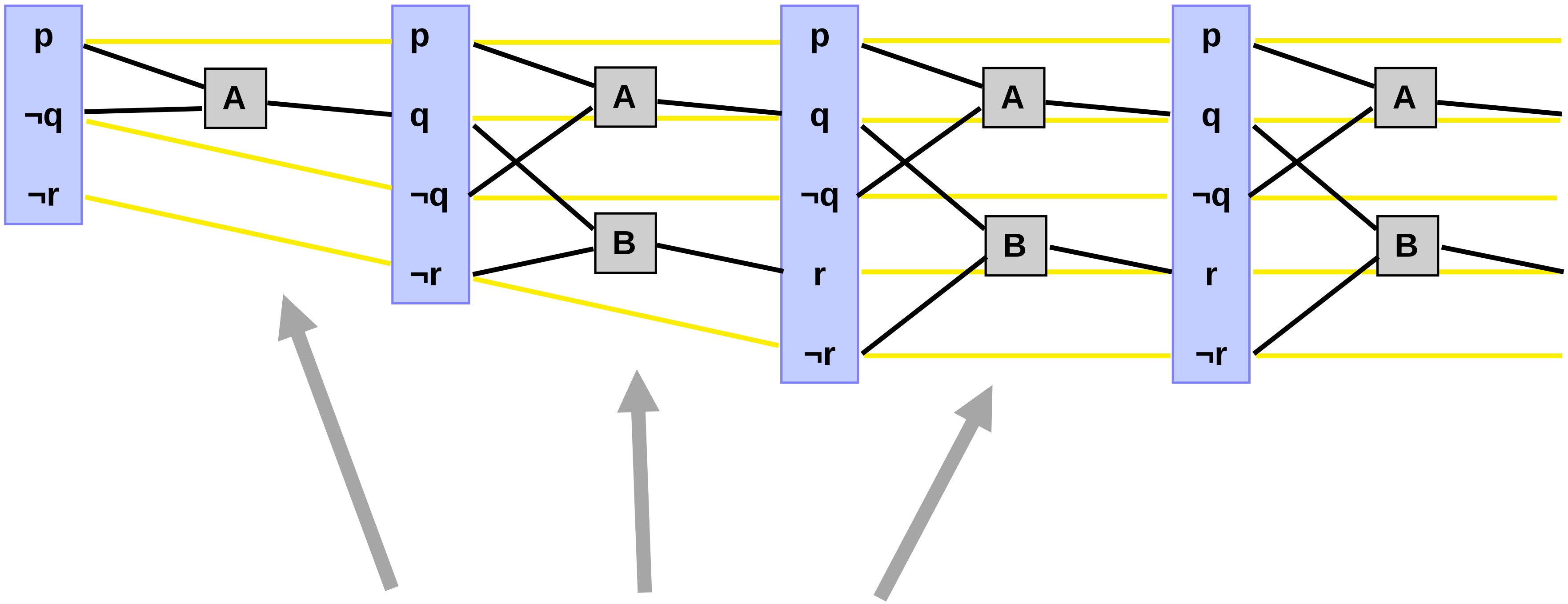
Observation 1



Propositions monotonically increase

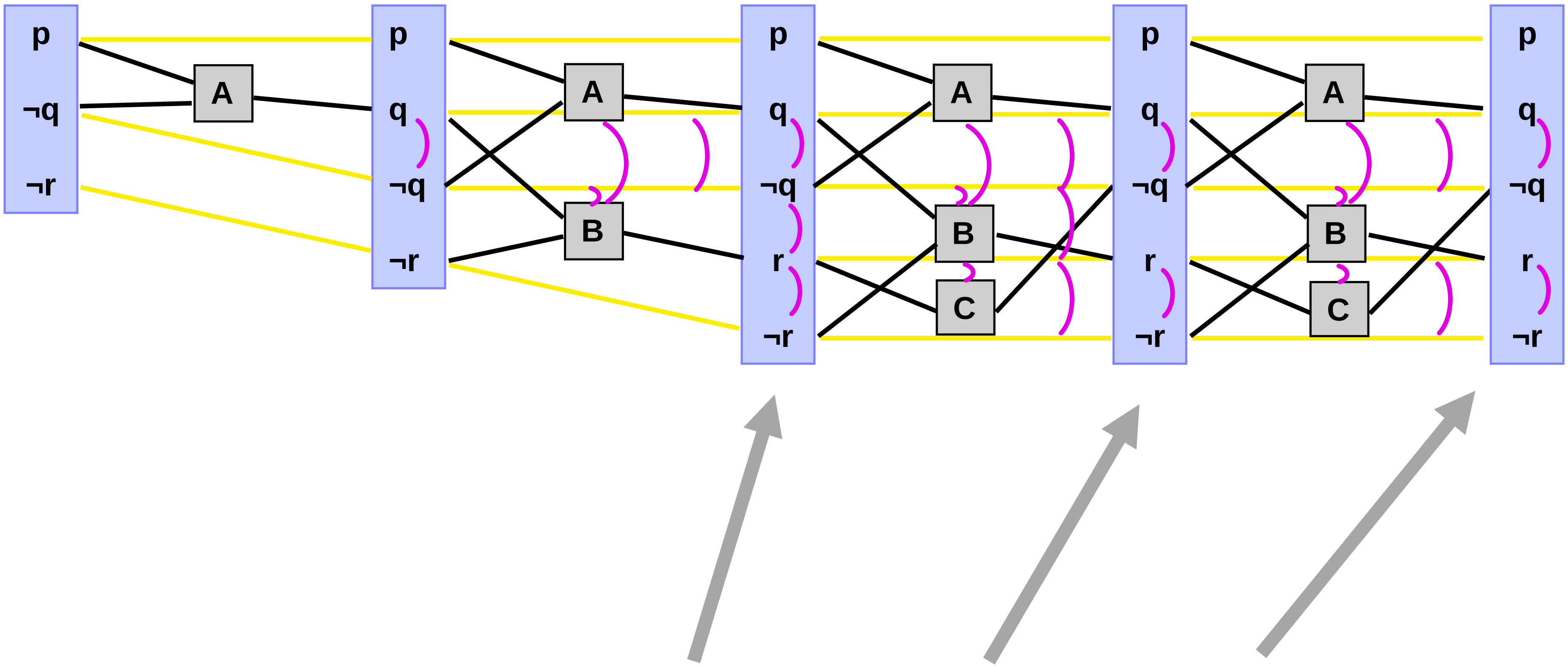
(always carried forward by no-ops)

Observation 2



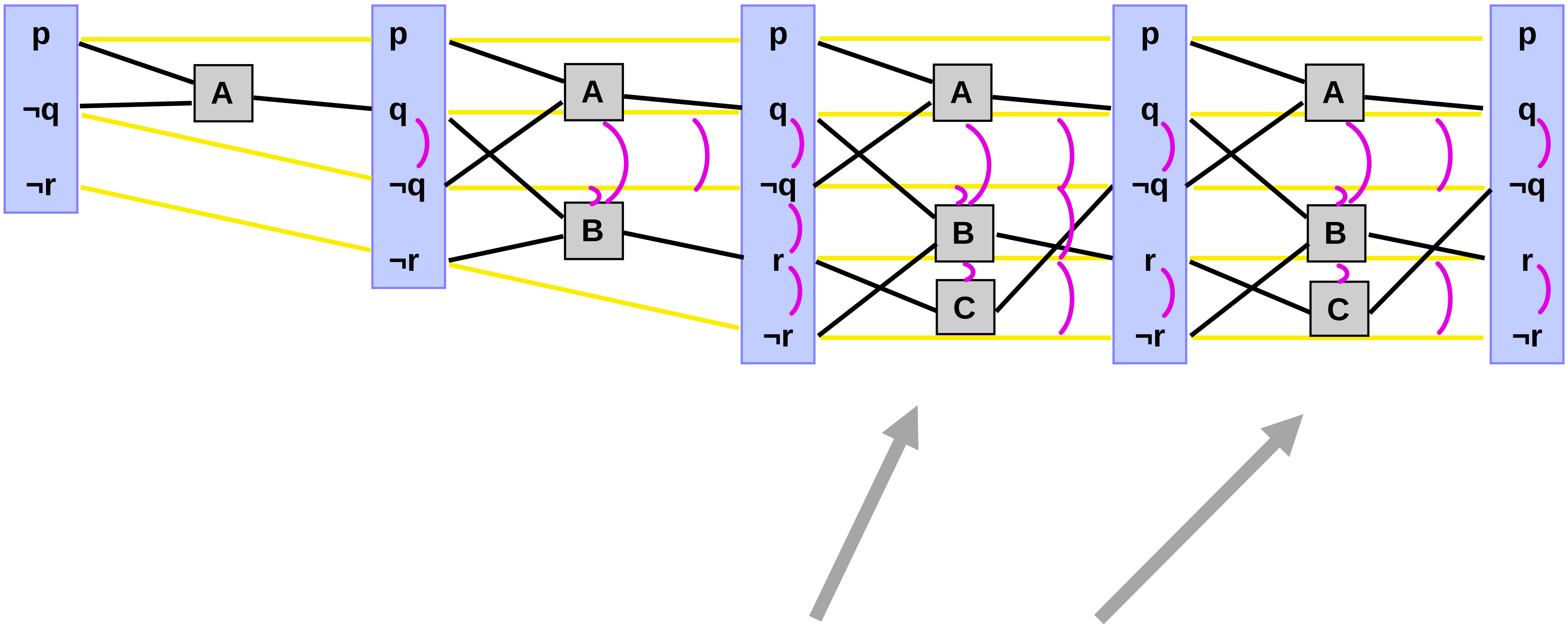
Actions monotonically increase

Observation 3



Proposition mutex relationships monotonically decrease

Observation 4



Action mutex relationships monotonically decrease

Observation 5

Planning Graph levels off（平稳）

- After some time steps all levels are identical
- Because it’s a finite space, the set of propositional literals never decreases and mutexes don’t reappear

Also provides a necessary and sufficient condition for termination of unsolvable problems:

- If planning graph has levelled off, yet goals are still unsatisfied

Valid plan

A *valid plan* is a subgraph of the planning graph where:

- Actions at the same level don't interfere or compete with each other (i.e. non-mutex actions)
- Each action's preconditions are made true by the plan
- Goals are satisfied

GraphPlan algorithm

Procedure GraphPlan:

- For k = 0, 1, 2, …

  <Graph expansion>:

  - Expand the planning graph level by level
    - If the planning graph levels off first, **fail**

  until all goals are reachable and not mutex

  <Solution extraction>:

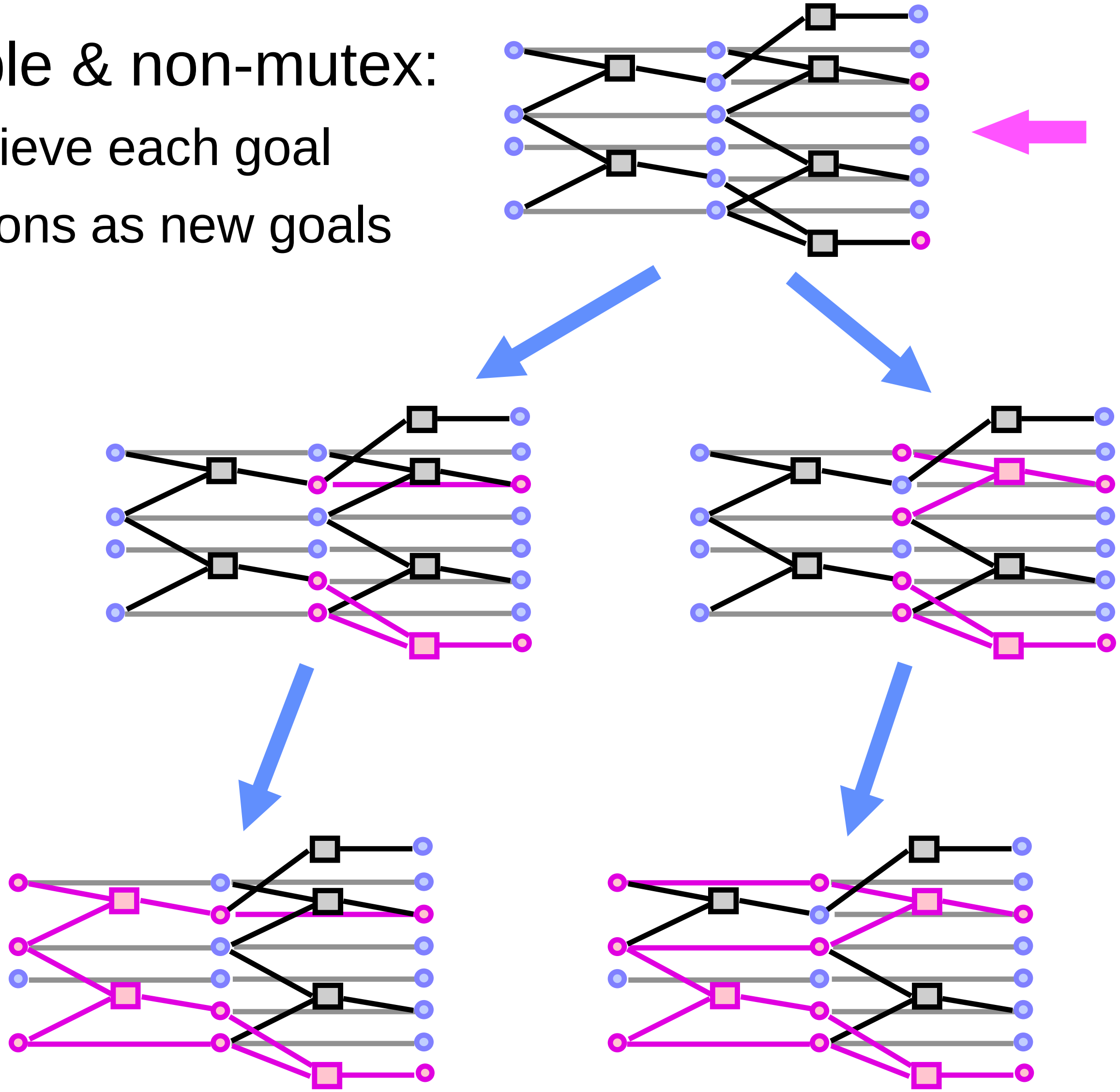
  - Backward search the planning graph for a valid plan
    - If a solution is found, then **return** *solution*

Searching for a solution plan

- Level-by-level backward chaining on the planning graph using mutex constraints
- Given a set of non-mutex goals at level *k*, identify a subset of non-mutex actions (including no-ops) at level *k*-1 to achieve current goals. The preconditions of these actions become new goals for level *k*-1

Searching for a solution plan

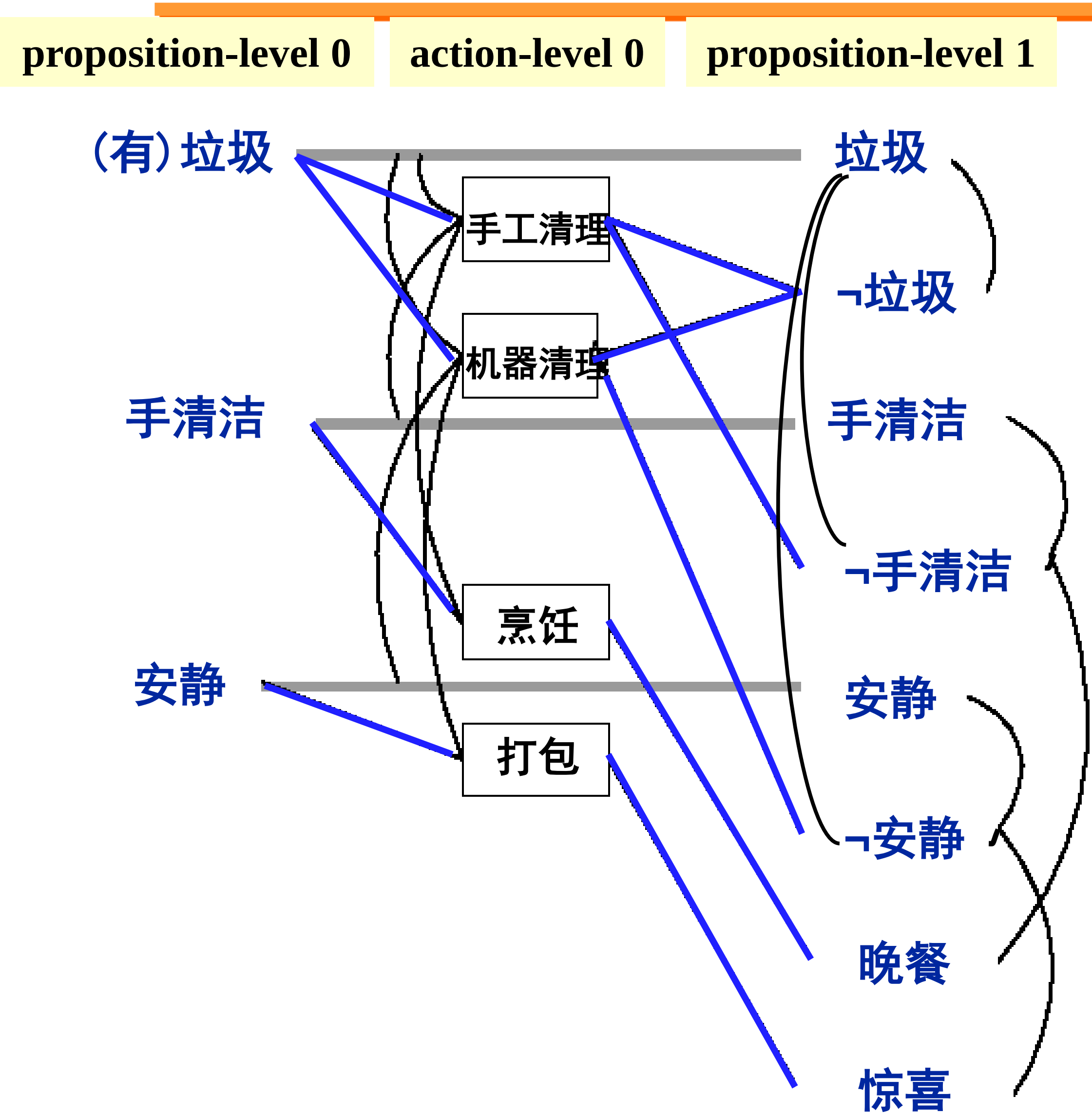
- If goals are reachable & non-mutex:
  - Choose action to achieve each goal
  - Add action preconditions as new goals



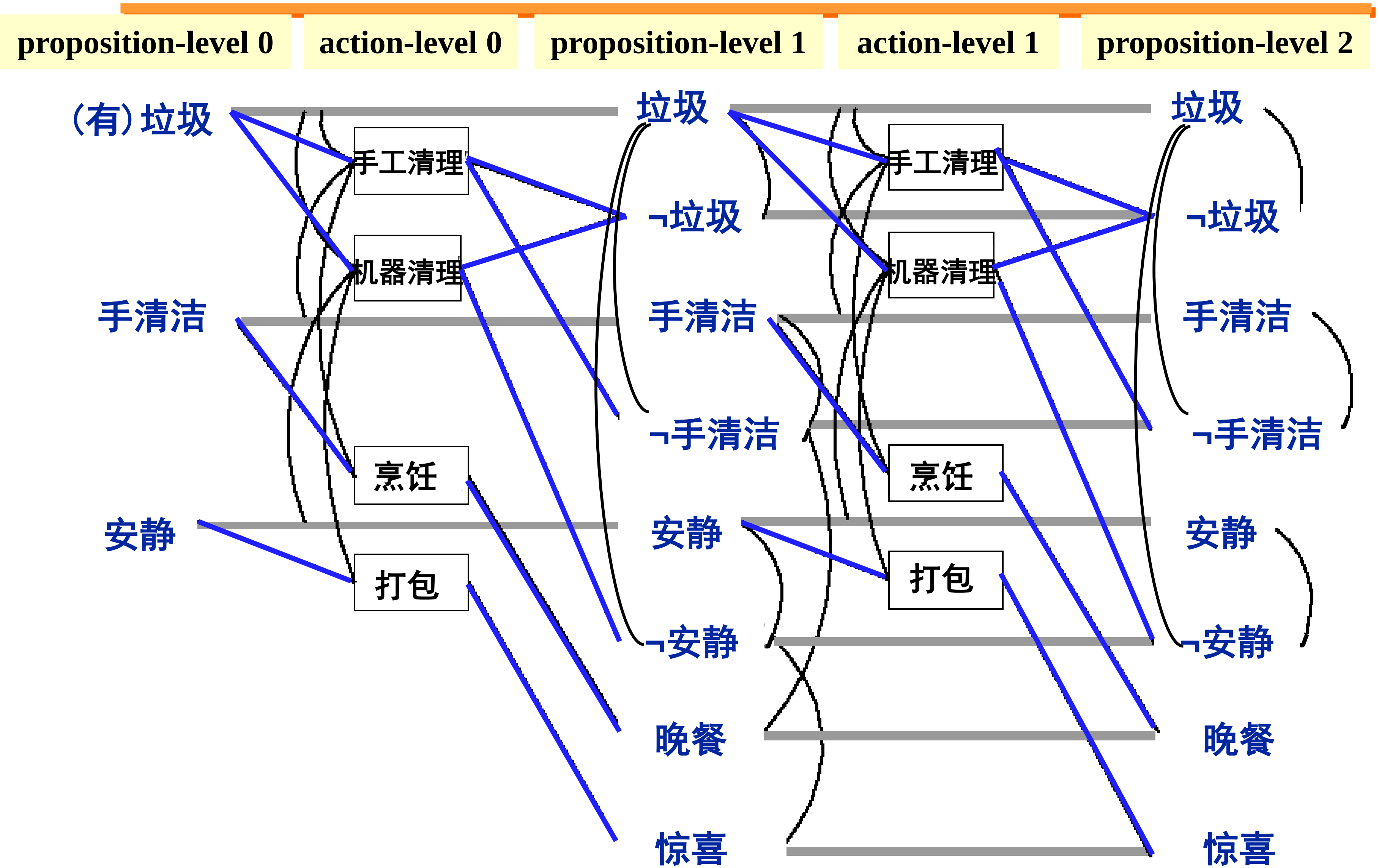
Dinner date example（为恋人准备惊喜晚餐）

- Initial state: （有）垃圾, 手清洁, 安静
- Goal: 晚餐, 惊喜, ¬垃圾
- Actions:
  - 烹饪: precondition (手清洁)
    effect (晚餐)
  - 打包: precondition (安静)
    effect (惊喜)
  - 手工清理: precondition (垃圾)
    effect（¬垃圾, ¬手清洁)
  - 机器清理: precondition (垃圾)
    effect ((¬垃圾, ¬安静)

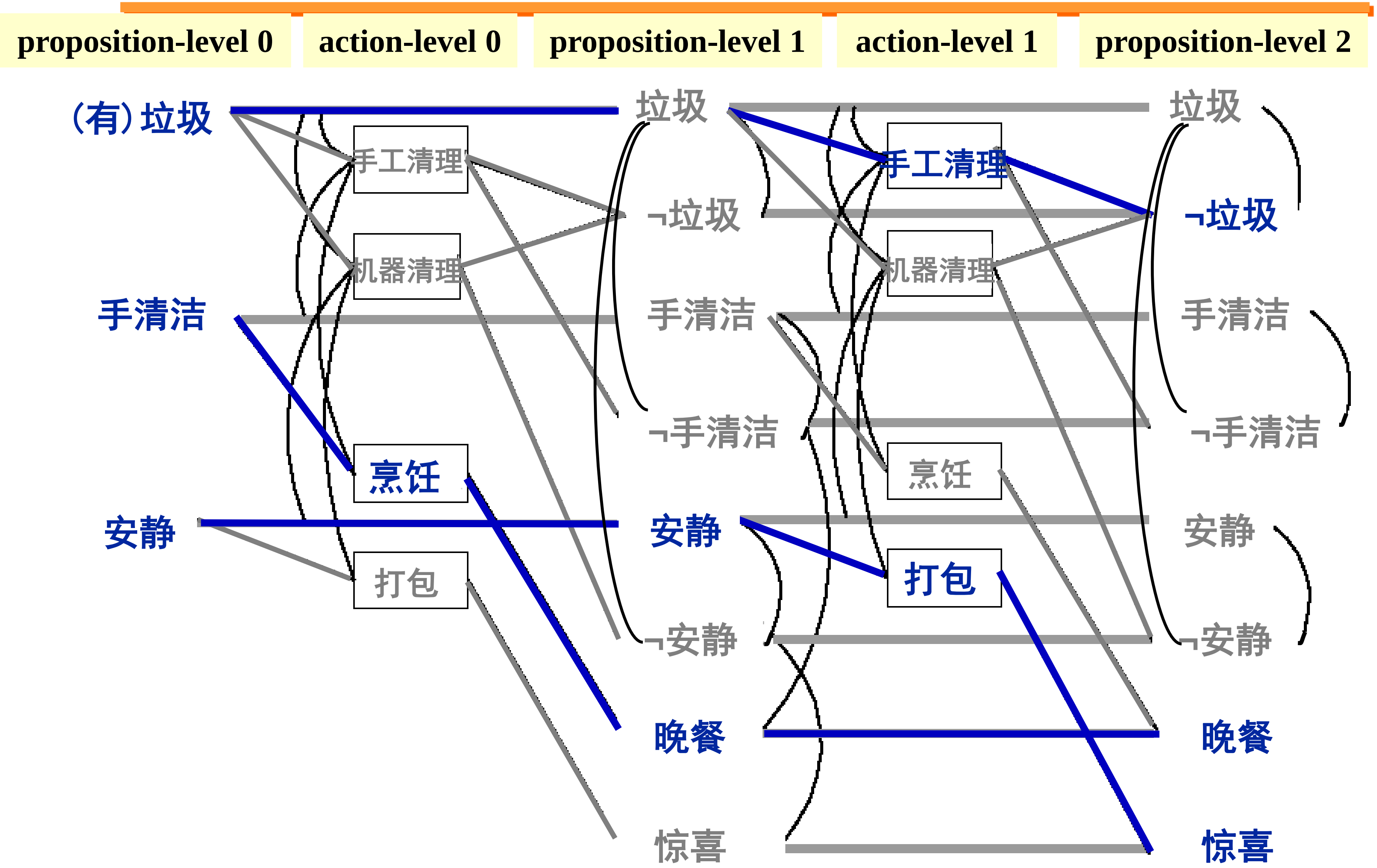
Dinner date example



Dinner date example



Dinner date example



GraphPlan exercise

- Initial state: (有) 食材, (有) 垃圾, 手清洁, 安静
- Goal: 晚餐, 惊喜, ¬垃圾
- Actions:
  - 烹饪: precondition (食材, 手清洁)
    effect (晚餐)
  - 打包: precondition (晚餐, 安静)
    effect (惊喜)
  - 手工清理: Precondition (垃圾)
    effect (¬垃圾, ¬手清洁)
  - 机器清理: precondition (垃圾)
    effect ((¬垃圾, ¬安静)

Properties of GraphPlan

Sound, *complete* and will terminate with failure if there is no plan

- Proved effective for solving *hard* planning problems
- Polynomial time graph construction
- Mutual exclusion (mutex) for pruning search
- Insensitivity to goal ordering
- Find "shortest parallel plan"

But work only for propositional planning problems (with no variables)

Comparison with plan-space planning

Advantage:

- The backward search of Graphplan (i.e. the hard part) only looks at actions in the planning graph
- Smaller search space than PSP, and thus faster

Disadvantage: 不能处理带变量、量词.

- To generate the planning graph, Graphplan creates a huge number of ground atoms
- Many of them may be irrelevant

For classical planning, advantage outweighs disadvantage

FF and extensions

Fast-forward planner [Hoffmann 01]

- A* search with heuristic values from:
  - *Relaxed* planning graph – only add effects

Extension to pure Strips operators [Koehler et al 97]:

- Disjunctive preconditions
- Negated preconditions
- Conditional effects
- Universal quantification

内容回顾

Outline

- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

Strips representation for actions:

Move-C-from-A-to-Table:

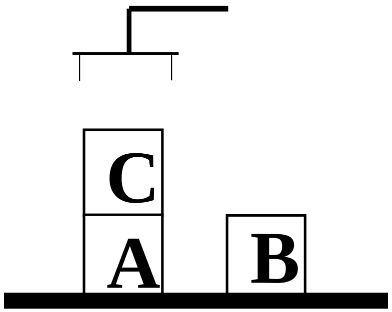
Precondition: (on C A), (clear C)

Effect:

add (on-table C)

add (clear A)

delete (on C A)



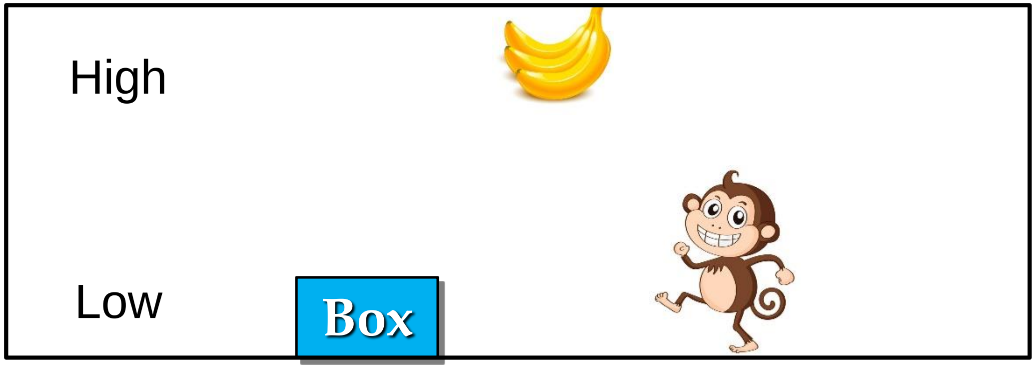
- The explicit effects are the only changes to the state

Means-ends analysis（手段目的分析）

Strips's problem solving takes **means-ends analysis**

- Search by reducing the difference between state and goals
- What *means* (operators) are available to achieve the desired *ends* (goal)

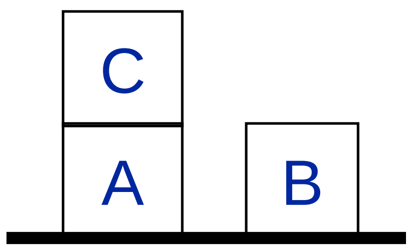
The monkey and bananas problem（猴子香蕉问题）



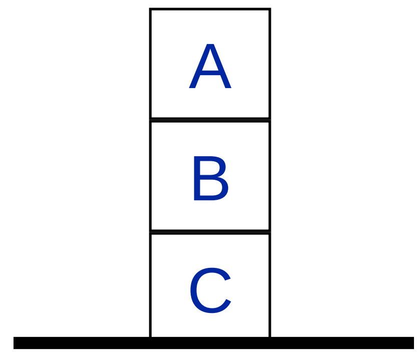
Blocks world example (Sussman anomaly)

- Strips uses *noninterleaved planner* (非交叉规划器), which cannot solve this example …

Initial:



Goal:



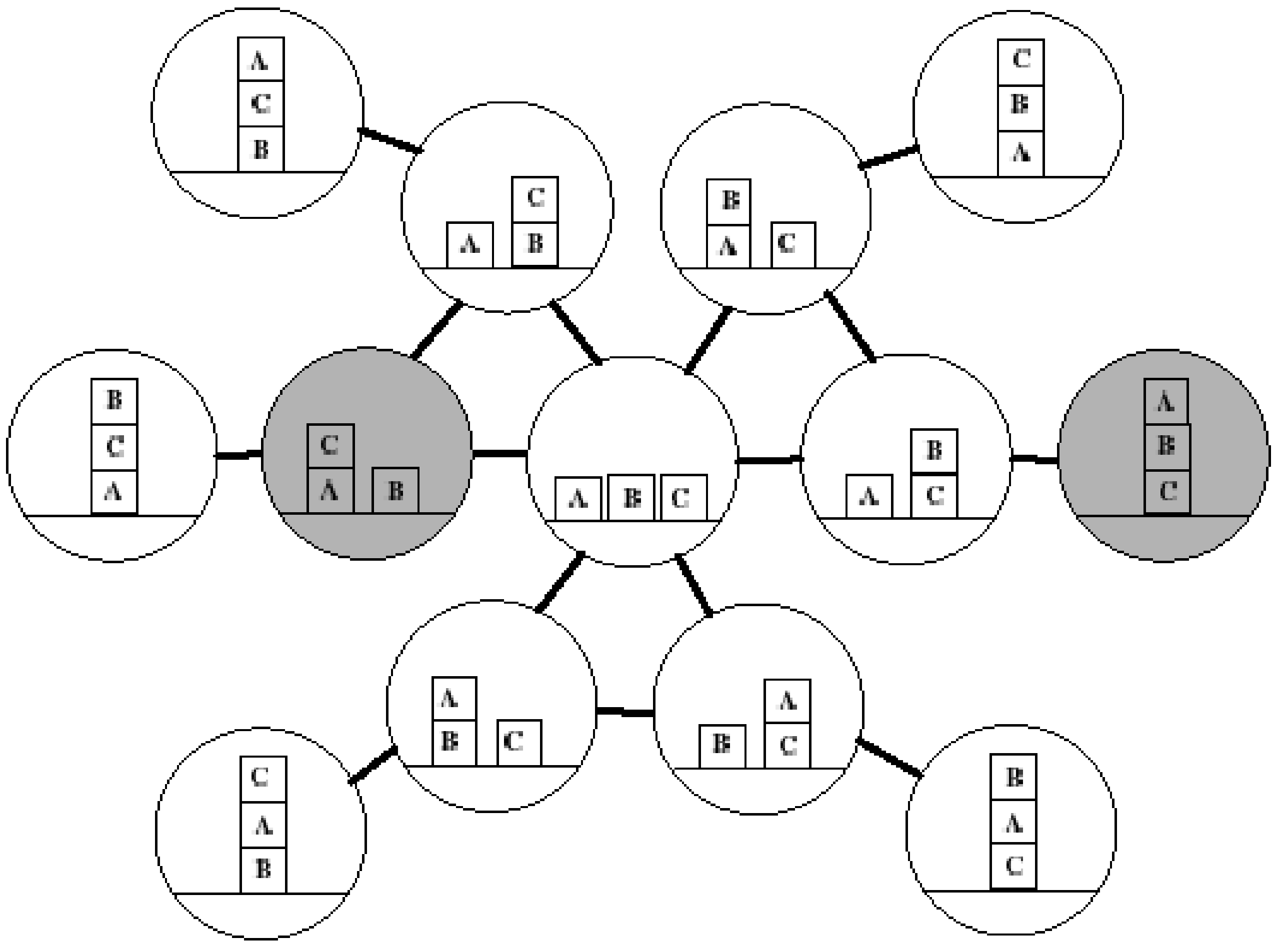
Initial state: on-table(A), on-table(B), on(C, A), clear(B), clear(C)

Goal: on(A,B), on(B,C) …

Outline

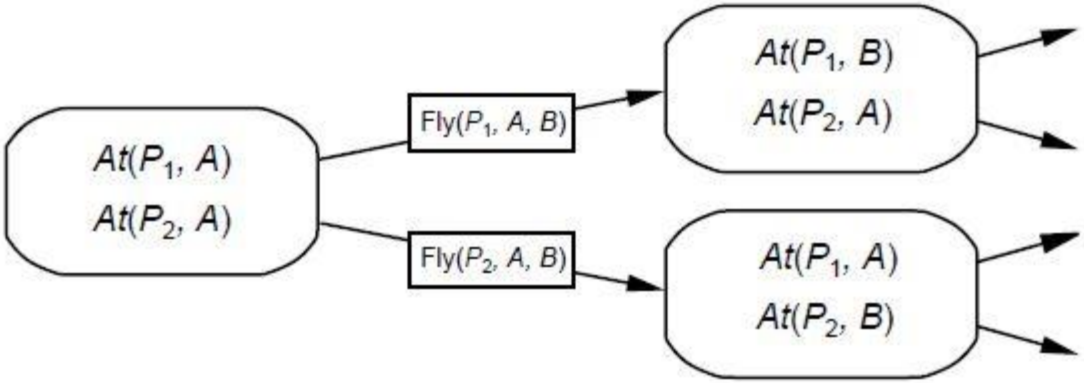
- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

Search space: Blocks world

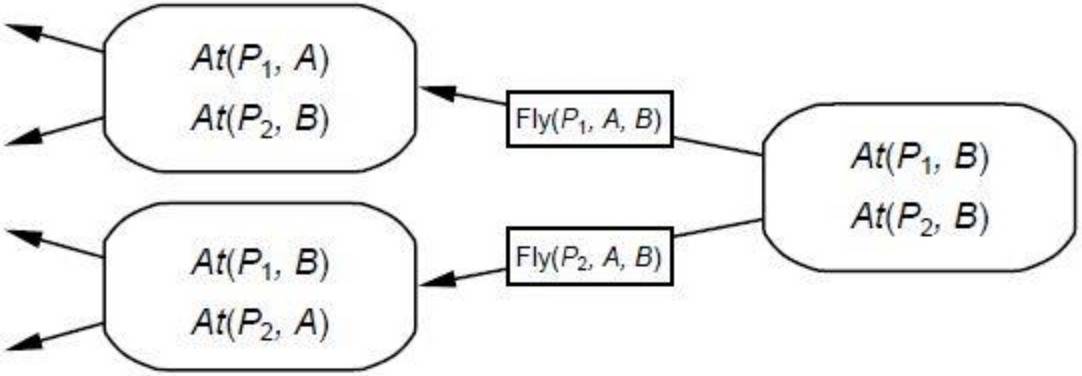


Planning algorithms

- Progression: Forward state-space search



- Regression: Backward state-space search



Progression vs regression

- Both algorithms are
  - *sound* (they always return a valid plan)
  - *complete* (if a valid plan exists they will find one)
- Complexity $O(b^n)$ worst-case
  - *where b = branching factor,*

    n = number of "choose" operators
- Regression: often smaller b, focused by goals
- Progression: full state space to compute heuristics

Outline

- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

Search the space of plans

Partial-Order Planning (POP): generates partial-order plans

- Nodes are partial plans
- Links are plan refinements
- Solution is a node (not a path)

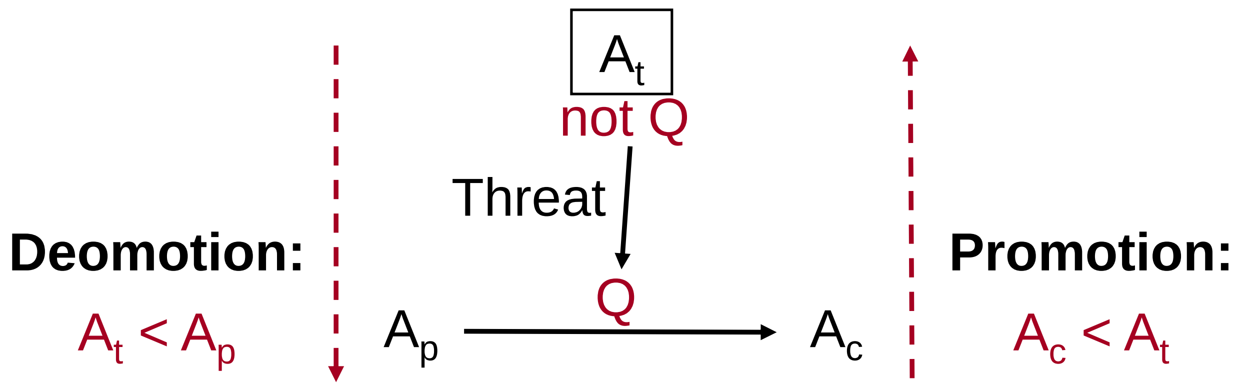
Follow the least commitment principle

- E.g. don’t commit to an order of actions until it is required

Partial plan representation

- Plan = (A, O, L), where
  - A: set of actions in the plan
  - O: *temporal orderings* between actions (a < b)
  - L: *causal links* linking actions via a literal
- Causal Link: $Ap \xrightarrow{Q} Ac$ （因果连接）

  Action Ac (consumer) has precondition Q that is established in the plan by Ap (producer)
- To protect causal link:



POP algorithm

POP((A, O, L), agenda, actions)

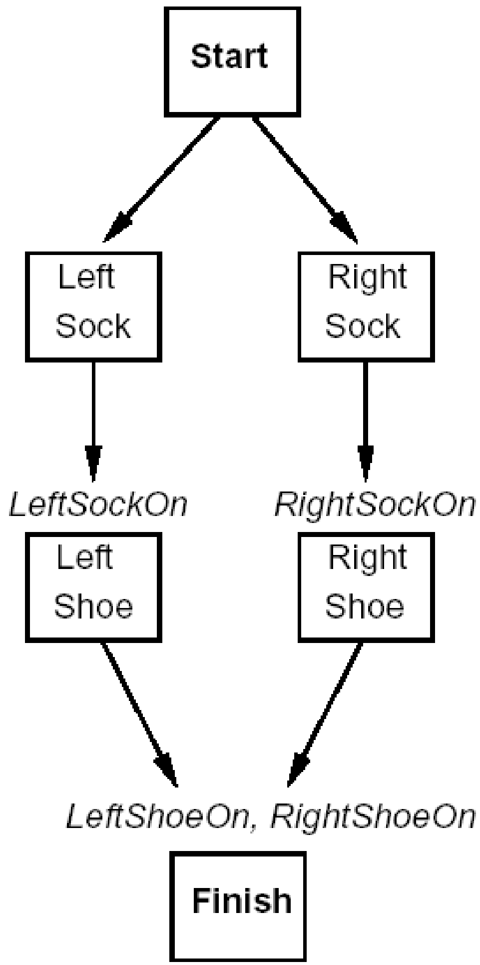
Initial plan: {*Start*, *Finish*} and preconditions in *Finish* as open conditions

1. If **agenda** is empty, then **return** (A, O, L) 结束条件
2. Pick (Q, A_{need}) from **agenda** (子)目标
3. **Choose** an action A_{add} that adds effect Q 动作选择
   - If no such action exists, **fail**
   - Add the link $A_{add} \xrightarrow{Q} A_{need}$ to **L** and the ordering $A_{add}<A_{need}$ to **O**
   - If A_{add} is new, add it to **A** 规划扩充
4. Remove (Q, A_{need}) from **agenda**. If A_{add} is new, for each of its preconditions P add (P, A_{add}) to **agenda** 更新(子)目标
5. For every action A_t in A that threatens any causal link $A_p \longrightarrow A_c$ in **L**
   - **Choose** to add $A_t<A_p$ or $A_c<A_t$ to **O**
   - If neither choice is consistent, **fail**
6. POP((A, O, L), **agenda**, actions)

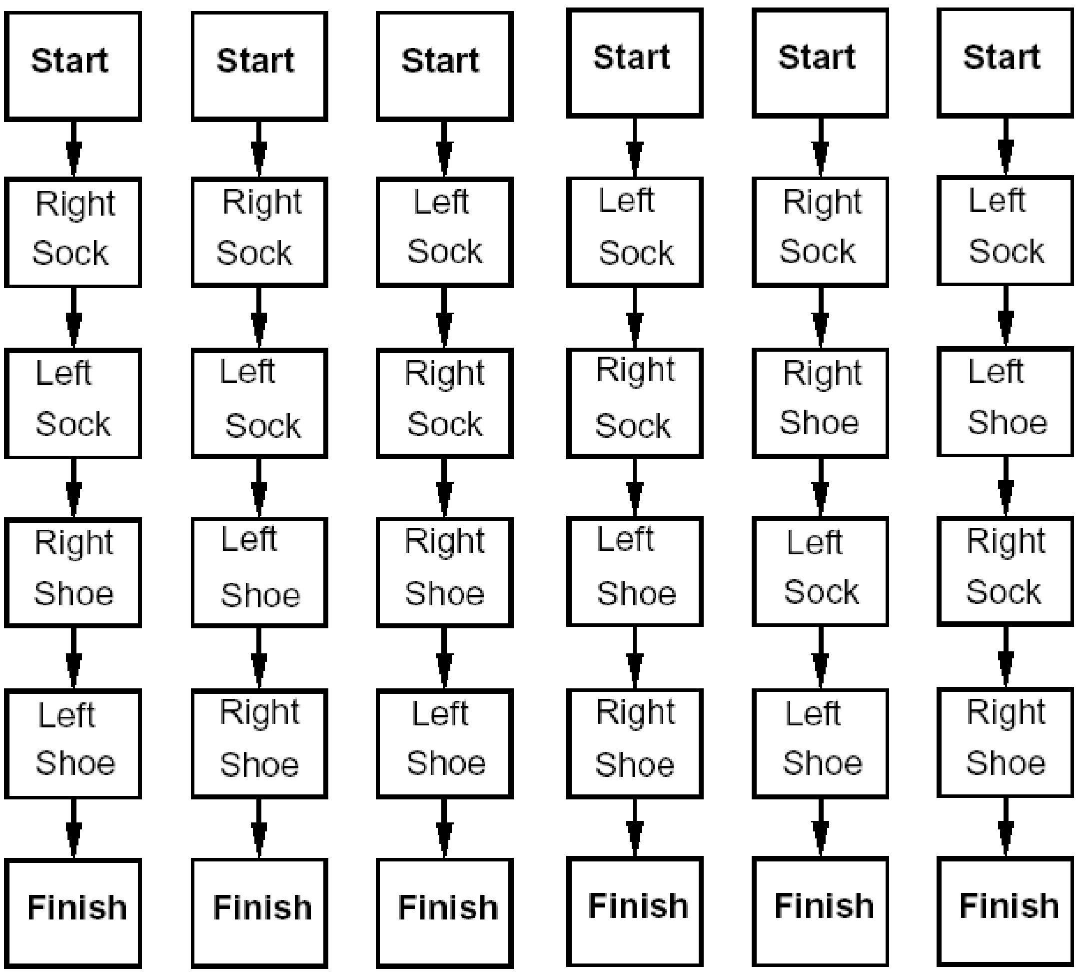
保护因果连接：
- 降级(Demotion): $\mathbf{A_t < A_p}$
- 升级(Promotion): $\mathbf{A_c < A_t}$

Total-order vs Partial-order plans

Partial Order Plan:



Total Order Plans: **（线性化）**



Properties of POP

POP is *sound* and *complete*

- Can be much faster than state-space planning, because of no need to backtrack over goal orderings (so less branching is required)
- Although it is more expensive per node and makes more choices than Regression, reduction in branching size often gains more
  - Larger *n* but smaller *b*

Flexibility gained by partial order

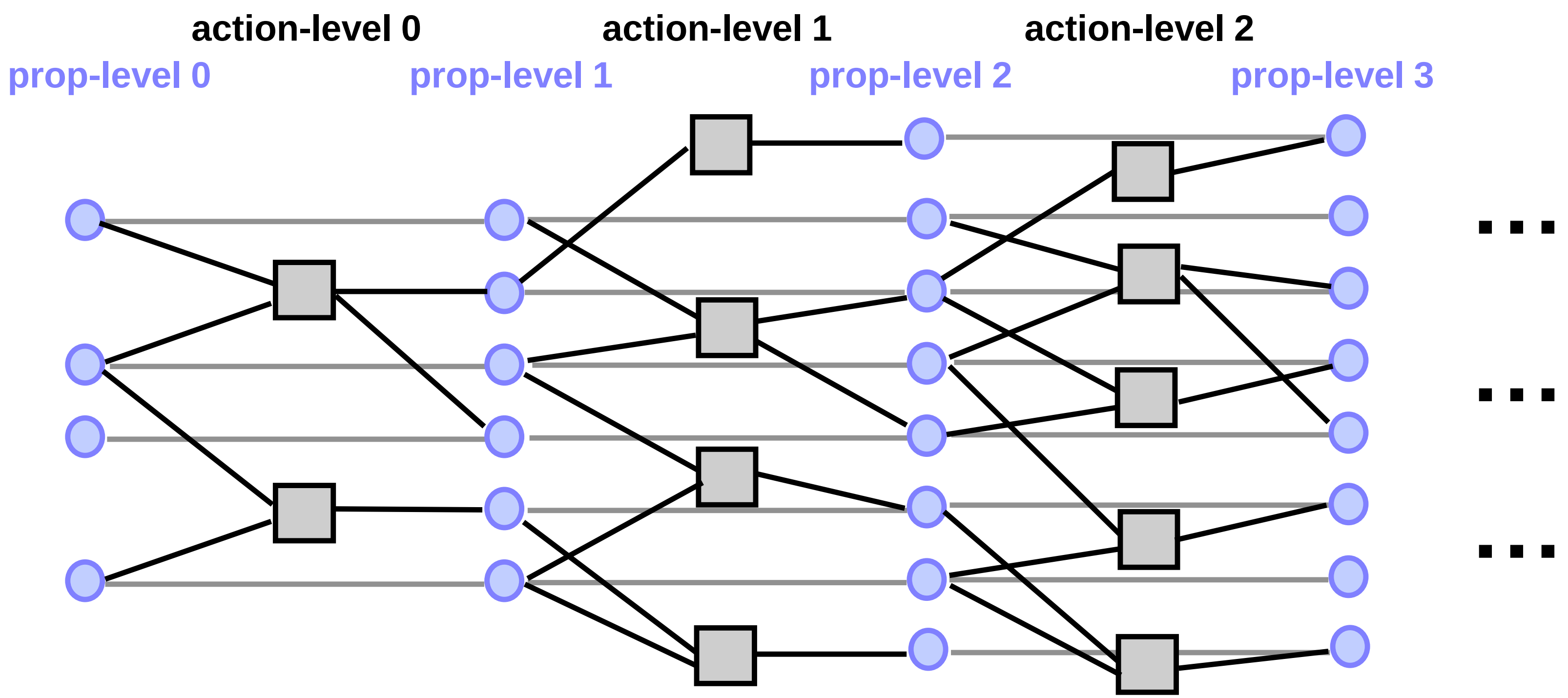
- Can be very useful to agent when the world fails to cooperate
- Make it easier to combine smaller plans into larger ones
  - Each small plan can reorder its actions to avoid conflict with the other plans

Outline

- STRIPS-like plan representation
- Planning with state-space search
- Partial-order planning (POP)
- Planning graphs (GraphPlan)

A planning graph（规划图）

- Consist of a sequence of **levels** that correspond to time steps in the plan, where level 0 is the initial state
- Each level contains a set of propositional literals and a set of actions, with edges connecting action preconditions and effects



Determining mutex relations（互斥关系）

- Two actions A and B are mutex at an action-level, if:
  - Interference: A (or B) disables a *precondition* or an *effect* of B (or A)
  - Competing needs: A and B have *mutex preconditions*
- Two propositions *p* and *q* are mutex at a proposition-level if
  - Negation: *p* (or *q*) is the *negation* of *q* (or *p*)
  - Inconsistent support: *all ways* of achieving them are *mutex*

    (i.e. all the actions that add *p* are mutual exclusion of all the actions that add *q*)

GraphPlan algorithm

Procedure GraphPlan:

- For k = 0, 1, 2, …

  <Graph expansion>:

  - Expand the planning graph level by level
    - If the planning graph levels off first, **fail**

  until all goals are reachable and not mutex

  <Solution extraction>:

  - Backward search the planning graph for a valid plan
    - If a solution is found, then **return** *solution*

Searching for a solution plan

- Level-by-level backward chaining on the planning graph using mutex constraints
- Given a set of non-mutex goals at level *k*, identify a subset of non-mutex actions (including no-ops) at level *k*-1 to achieve current goals. The preconditions of these actions become new goals for level *k*-1

A *valid plan* is a subgraph of the planning graph where:

- Actions at the same level don't interfere or compete with each other (i.e. non-mutex actions)
- Each action's preconditions are made true by the plan
- Goals are satisfied

Properties of GraphPlan

Sound, *complete* and will terminate with failure if there is no plan

- Proved effective for solving *hard* planning problems
- Polynomial time graph construction
- Mutual exclusion (mutex) for pruning search
- Insensitivity to goal ordering
- Find "shortest parallel plan"

But work only for propositional planning problems (with no variables)

References

- R. Fikes and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3-4), 1971
- D. Weld. An Introduction to Least-Commitment Planning. *AI Magazine*, 15(4), 1994 (POP)
- J. S. Penberthy and D. Weld. UCPOP: A Sound, Complete, Partial-Order Planner for ADL. *Proceedings of KR*, 1992
- A. Blum and M. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2), 1997 (GraphPlan)
- J. Hoffmann. FF: The Fast-Forward Planning System. *AI Magazine*, 22(3), 2001

Resources

- GraphPlan Planner:

  https://en.wikipedia.org/wiki/Graphplan (External links)

- UCPOP Planner:

  https://www.swmath.org/software/20687

- Action Description Language (ADL):

  https://handwiki.org/wiki/Action_description_language

- Planning Domain Definition Language (PDDL):

  https://planning.wiki/guide/whatis/pddl

End.