# Planning Graph Techniques

Dra. Mª Dolores Rodríguez Moreno

Universidad de Alcalá

ISG

# Objectives

**Specific Objectives**

- Graph-based Planners (GP) techniques

**Source**

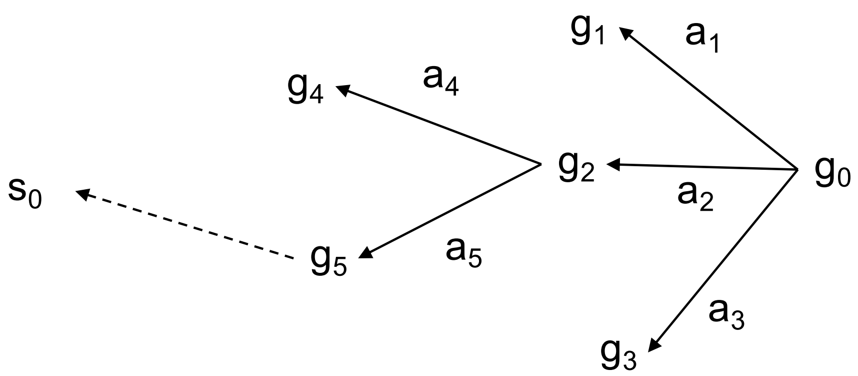
- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. Chapter 10. (3rd Edition). Ed. Pearsons
- Dana Nau's slides for Automated Planning. Licensed under License https://creativecommons.org/licenses/by-nc-sa/2.0/
- Blum & Furst (1997). Fast planning through planning graph analysis. AI. 90:281-300.

# Outline

- **Motivation**
- Procedure
- GP-based planners
- Baking example
- State reachability
- Comparison with PSP
- GP planners
- Todo example

# Motivation (I)

- A big source of inefficiency in search algorithms is the *branching factor* (i.e. the number of children of each node)
- A backward search may try lots of actions that can't be reached from the initial state



# Motivation (II)

- Reduce branching factor, how?
- First 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
- Then do a modified version of the original search
  - Restrict its search space to include only those actions that occur in solutions to the relaxed problem

# Outline

- Motivation
- **Procedure**
- GP-based planners
- Baking example
- State reachability
- Comparison with PSP
- GP planners
- Todo example

## Procedure: graph expansion

- for $k = 0, 1, 2, \ldots$
  - *Graph expansion:*
    - create 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
    - do *solution extraction:*
      - backward search, modified to consider only the actions in the planning graph
      - if we find a solution, then return it

relaxed problem

# Procedure: solution extraction

procedure Solution-extraction($g, j$)
    if $j$=0 then return the solution
    for each literal $l$ in $g$
      non-deterministically choose an action
        to use in state $s_{j-1}$ to achieve $l$
    if any pair of chosen actions are mutex
      then backtrack
    $g'$:= {the preconditions of the chosen actions}
    Solution-extraction($g', j-1$)
end

The set of goals we are trying to achieve

The level of the state $s_j$

A real action or a maintenance action

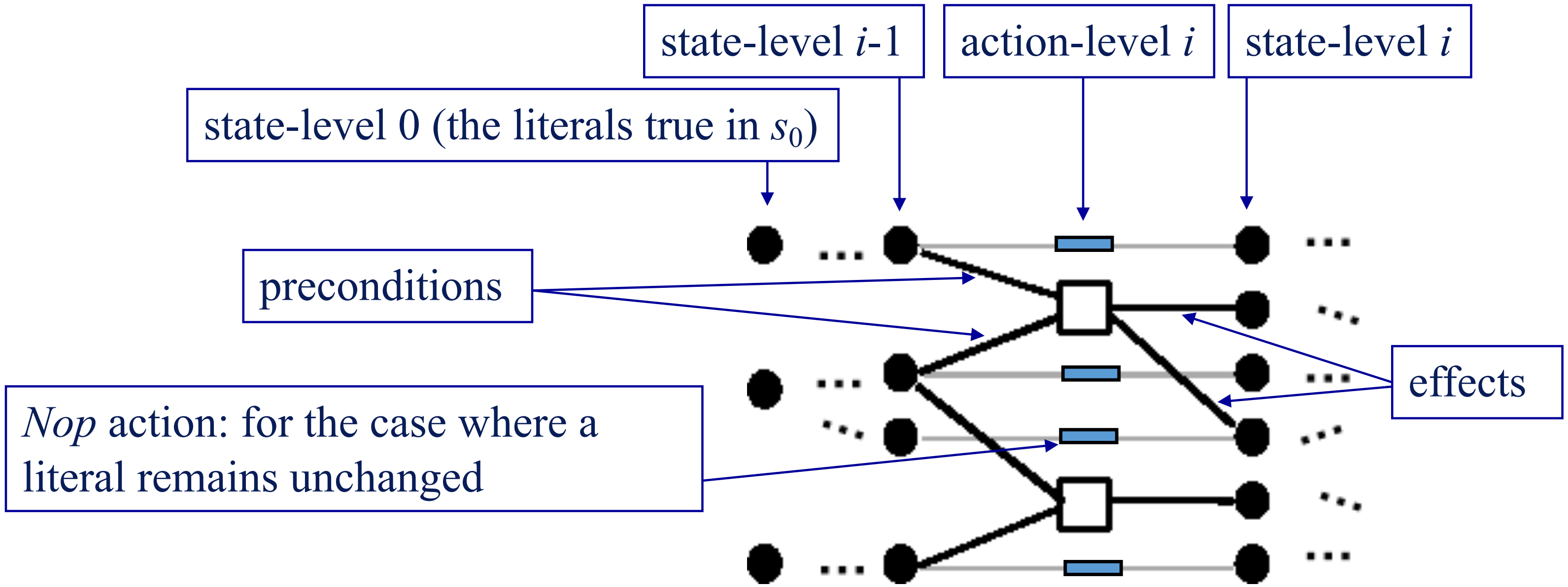
# Outline

- Motivation
- Procedure
- **GP-based planners**
- Baking example
- State reachability
- Comparison with PSP
- GP planners
- Todo example

# Graph-based Planners (I)

- The search structure is based on a Planning Graph
- The graph is directed and layered:
  - 2 types of nodes:
    - Proposition nodes: even levels (initial state → 0)
    - Action nodes: odd levels
  - 3 types of arcs: represent relationships between actions and propositions:
    - Added
    - Deleted
    - Nop

# Graph-based Planners (II)



# Graph-based Planners (III)

- GP algorithm works in two alternating phases
  - Expands (add layers) the planning graph until the last proposition layer satisfies the goal condition
  - Try to extract a valid plan (backtracking) from the planning graph

- If unsuccessful continues with the former phase, the planning graph is expanded again

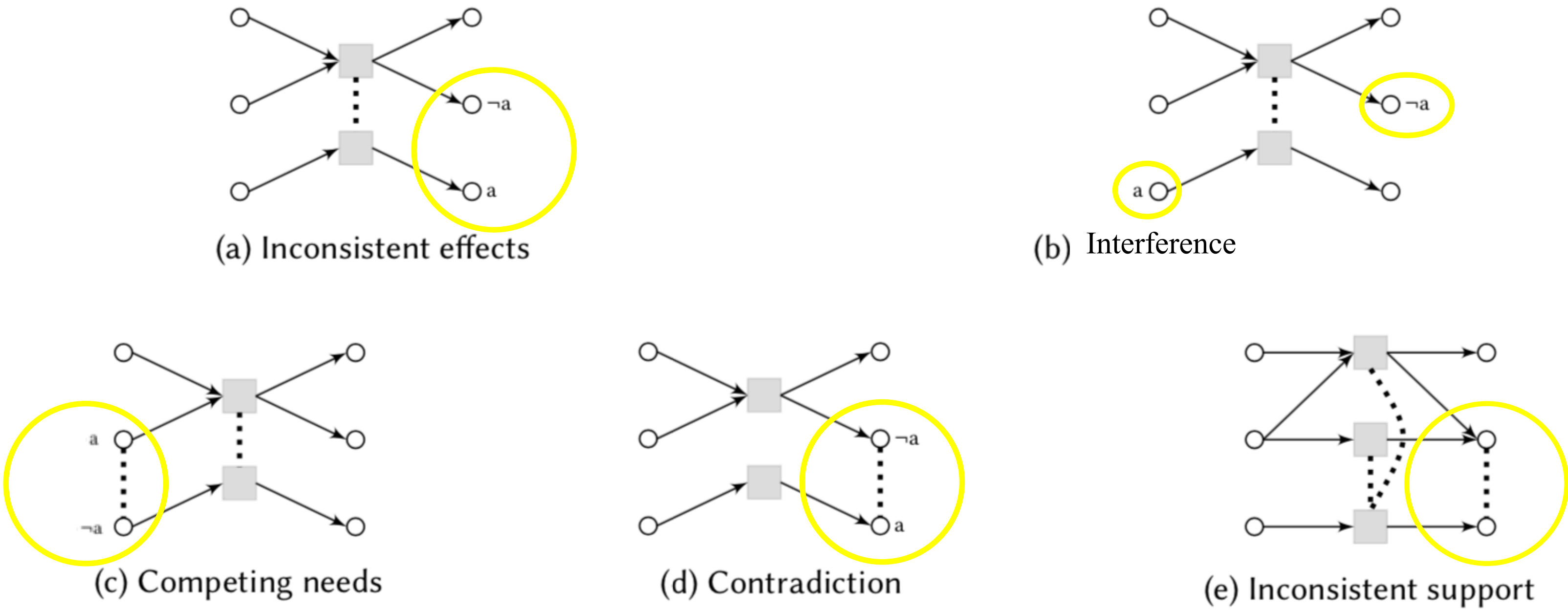
# Graph-based Planners (IV)

- It is necessary to develop a reachability analysis to reduce the set of actions that are not supported in each layer
- Compatibility inferring mutual exclusion relations between incompatible **actions** is performed (mutex)
  - Have opposite effects
  - Incompatible preconditions
  - The effect of one action is the opposite of another
- Mutex between incompatible **propositions**: negated literals or all actions that can achieve them are mutex in the previous step

# Graph-based Planners (V)

- Two actions at the same action-level are mutex if
  - *Inconsistent effects:* an effect of one negates an effect of the other
  - *Interference:* one deletes a precondition of the other
  - *Competing needs:* **they have mutually exclusive preconditions**
- Otherwise they don't interfere with each other (may appear in solution)
- Two literals at the same state-level are mutex if
  - *Inconsistent support:* one is the negation of the other (contradiction), **or all ways of achieving them are pairwise mutex**

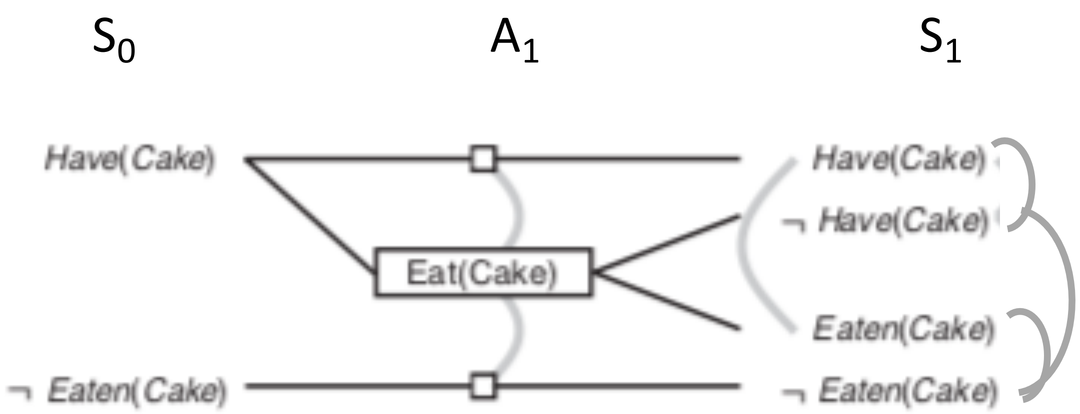
# Graph-based Planners (VI)

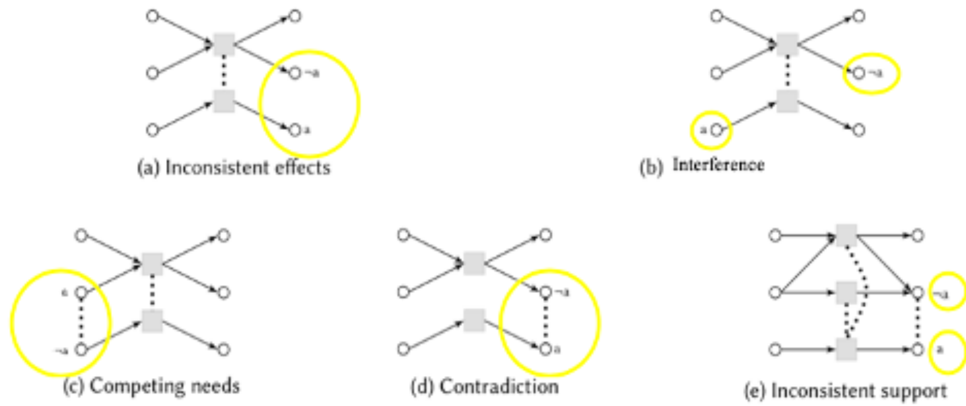


# Outline

- Motivation
- Procedure
- GP-based planners
- **Baking example**
- State reachability
- Comparison with PSP
- GP planners
- Todo example

# Example




```
(:action Eat
   :parameters (?cake)
   :precondition (and (have ?cake))
   :effect (and (eaten ?cake) (not (have ?cake)))
(:action Bake
   :parameters (?cake)
   :precondition (not (have ?cake))
   :effect (and (have ?cake))))
```

Initial State

Have (cake) & (not (Eaten(cake))

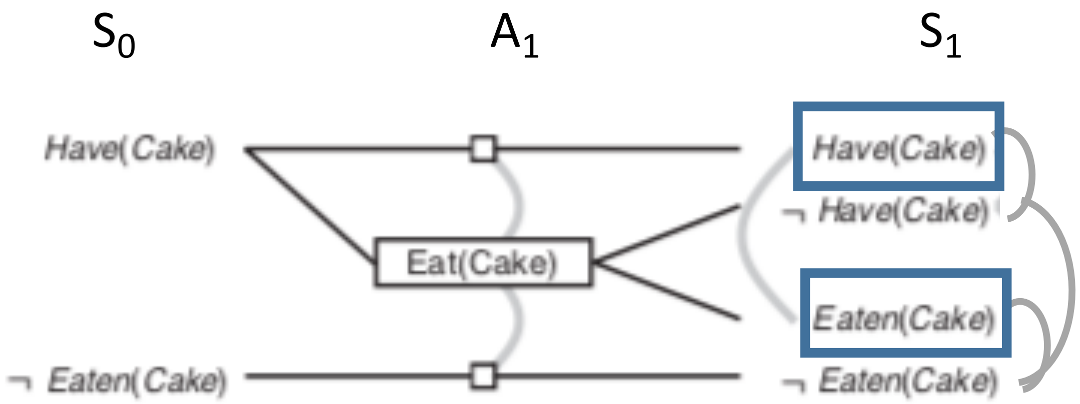
Goal

Eaten (cake) and Eaten (cake)

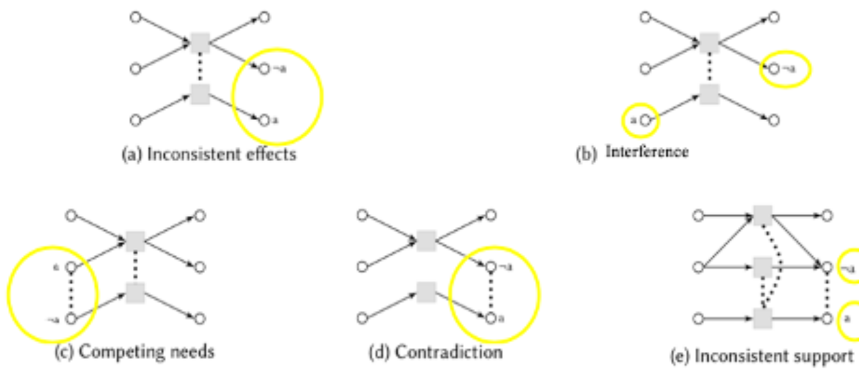
# Example: mutex

- *Inconsistent effects:*
  - Eat(C) and no-op Have(C) because they disagree on the effect Have(C)
  - Eat(C) and no-op $\neg$ Eaten(C) because they disagree on the effect $\neg$ Eaten(C)
- *Contradiction*:
  - Have(C) and not Have(C),
  - Eaten(C) and not Eaten(C)
- *Inconsistent support*:
  - Have(C) and Eaten(C) are mutex because the only way of achieving Have(C), the no-op action, is mutex with the only way of achieving Eaten (C).
  - The same between *not Have(C)* and *not Eaten(C)*

# Example

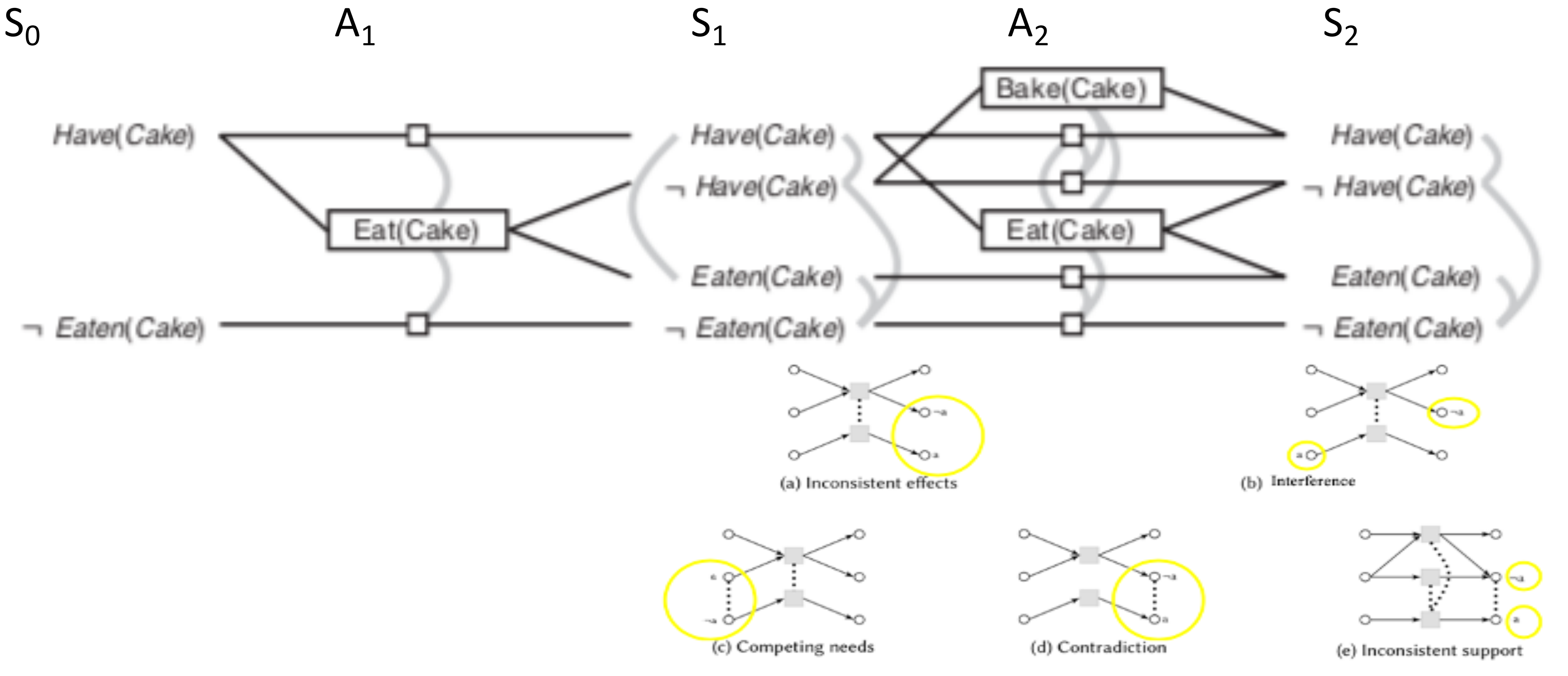



(:action Eat
    :parameters (?cake)
    :precondition (and (have ?cake))
    :effect (and (eaten ?cake) (not (have ?cake)))
(:action Bake
    :parameters (?cake)
    :precondition (not (have ?cake))
    :effect (and (have ?cake))))




Initial State
    Have (cake) & (not (Eaten(cake))
Goal
    Eaten (cake) and Eaten (cake)

# Example

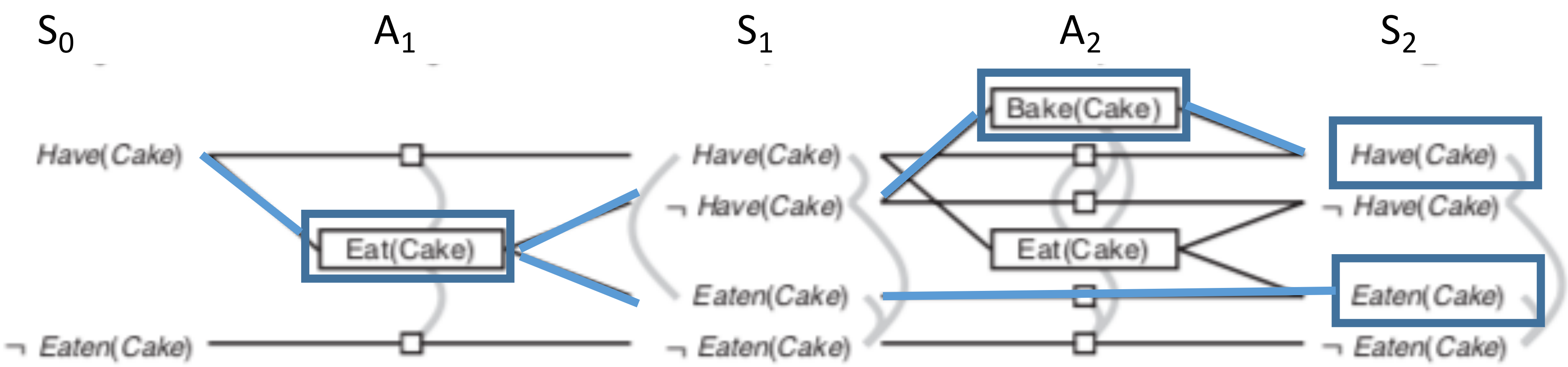


$S_0$
$A_1$
$S_1$
$A_2$
$S_2$
Have(Cake)
¬ Eaten(Cake)
Eat(Cake)
Have(Cake)
¬ Have(Cake)
Eaten(Cake)
¬ Eaten(Cake)
Bake(Cake)
Eat(Cake)
Have(Cake)
¬ Have(Cake)
Eaten(Cake)
¬ Eaten(Cake)
(a) Inconsistent effects
(b) Interference
(c) Competing needs
(d) Contradiction
(e) Inconsistent support

# Example: mutex

- *Competing needs:*
  - Bake(C) and Eat(C) are mutex because they compete on the value of the Have(C) precondition
- *Inconsistent effects:*
  - Eat(C) and no-op Have(C) because they disagree on the effect Have(C)
  - Eat(C) and no-op ¬ Eaten(C) because they disagree on the effect ¬ Eaten(C)
  - No-op ¬ Eaten(C) and Eaten(C) (the same with Hand(C))
  - Bake(C) and no-op ¬ Have(C) because they disagree on the effect Have(C)
- *Contradiction*: Have(C) and not Have(C), Eaten(C) and not Eaten(C)
- Note that in S2 Have (C) and Eaten(C) are not mutex since we can achieve Have(C) with Bake

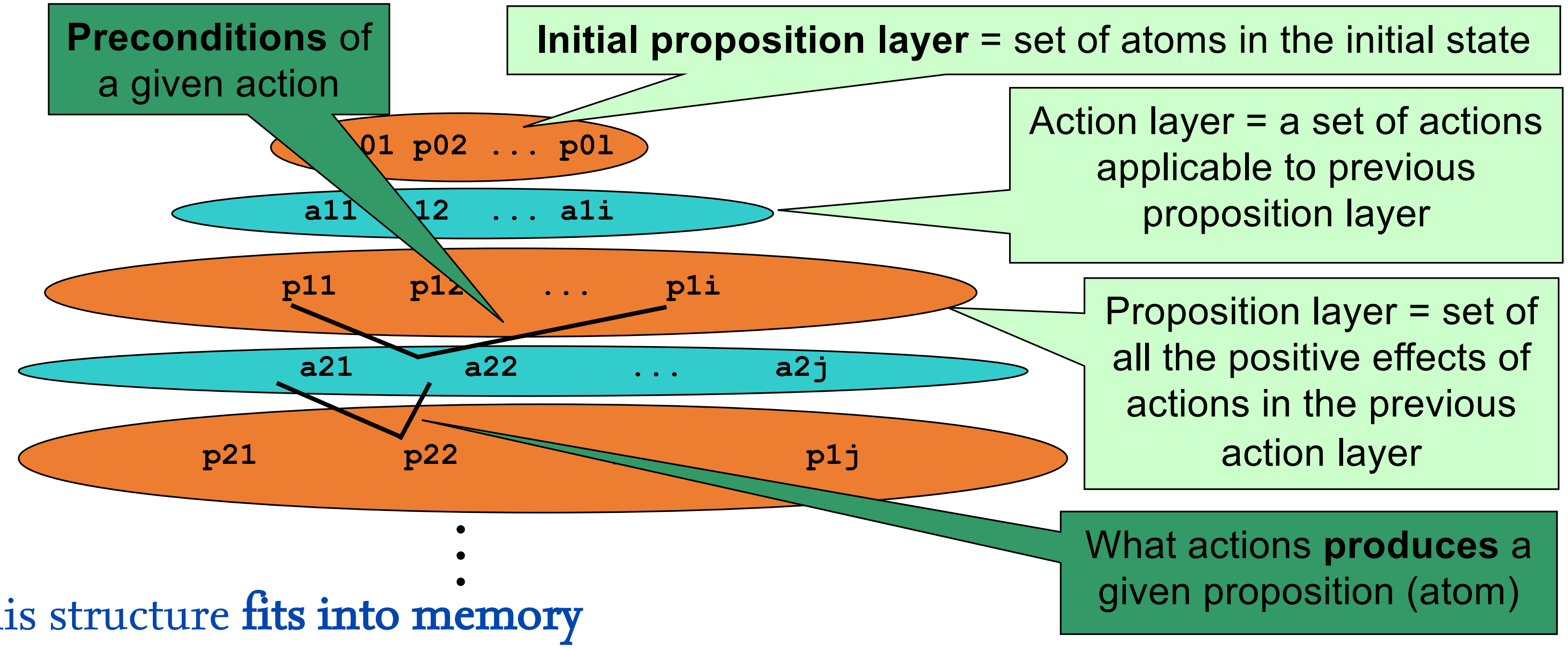
# Example: Solution Extraction



# Outline

- Motivation
- Procedure
- GP-based planners
- Baking example
- **State reachability**
- Comparison with PSP
- GP planners
- Todo example

# State reachability




- this structure **fits into memory**
  - every **proposition** knows its origin in the previous action layer
  - every **action** knows its precondition in the previous proposition layer

# Outline

- Motivation
- Procedure
- GP-based planners
- Baking example
- State reachability
- **Comparison with PSP**
- GP planners
- Todo example

# Comparison with Plan-Space Planning

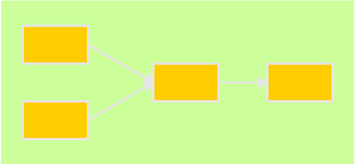
- Advantage:
    - The backward-search part of GP—which is the hard part—will only look at the actions in the planning graph
    - Smaller search space than PSP; thus faster
- Disadvantage:
    - To generate the planning graph, GP creates a huge number of ground atoms
    - Many of them may be irrelevant
- Can alleviate (but not eliminate) this problem by assigning data types to the variables and constants
    - Only instantiate variables to terms of the same data type
- For classical planning, the advantage outweighs the disadvantage
    - GP solves classical planning problems much faster than PSP

# Outline

- Motivation
- Procedure
- GP-based planners
- Baking example
- State reachability
- Comparison with PSP
- **GP planners**
- Todo example

# GP Planners

- A big number of descendent
  - SGP: contingent planner
  - TGP: includes temporal reasoning
  - IPP: allows resource reasoning
  - TPSYS: combines GP & TGP
  - SAPA: uses a set of heuristics based on distances to control the search
  - STAN: extracts admissible heuristics from a domain analysis tool called TIM
  - LPG: combines GP & SAT
  - ...
- Output



# Outline

- Motivation
- Procedure
- GP-based planners
- Baking example
- State reachability
- Comparison with PSP
- GP planners
- **Todo example**

# Example II

- Suppose you want to prepare dinner

  $s_o$ = {garbage, cleanHands, quiet}

  $g$ = {dinner, present, ¬garbage}

| Action | Preconditions | Effects |
|---|---|---|
| cook() | cleanHands | dinner |
| wrap() | quiet | present |
| carry() | *none* | ¬garbage, ¬cleanHands |
| dolly() | *none* | ¬garbage, ¬quiet |

- Specify one case of mutex and the level when it occurs:
  - Interference
  - Inconsistent support

# Example II

- Suppose you want to prepare dinner

  $s_o$ = {garbage, cleanHands, quiet}

  $g$ = {dinner, present, ¬garbage}

Let's do a Socrative!!!

| Action | Preconditions | Effects |
|---|---|---|
| cook() | cleanHands | dinner |
| wrap() | quiet | present |
| carry() | *none* | ¬garbage, ¬cleanHands |
| dolly() | *none* | ¬garbage, ¬quiet |

- Specify one case of mutex and the level when it occurs:
  - Interference
  - Inconsistent support