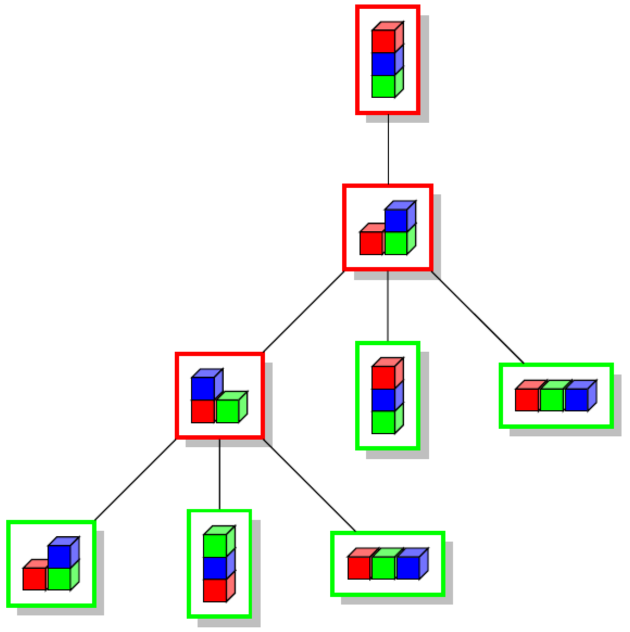
The Big Three Planning Approaches

- Of the many planning approaches, three techniques stand out:
- 1 Graph/SAT Planning
- 2 Symbolic Search Planning
- 3 Heuristic State-Space Search

The Big Three Planning Approaches

- Of the many planning approaches, three techniques stand out:
- 1 Graph/SAT Planning
- 2 Symbolic Search Planning
- 3 Heuristic State-Space Search

State-Space Search in a Nutshell



Closed node

Open node

Which open node should we select for expansion?

GraphPlan: Basic idea

- ➢Construct a graph that encodes constraints on possible plans
- ➢Use this “planning graph” to constrain search for a valid plan
- ➢Planning graph can be built for each problem in a relatively short time
- ➢Finds “shortest” plans (makespan)
- ➢Sound, complete, and will terminate with failure if there is no plan

Planning graph

- Directed, leveled graph with alternating layers of nodes
- Odd layers ("state levels") represent candidate propositions that could possibly hold at step *i*
- Even layers ("action levels") represent candidate actions that could possibly be executed at step *i*, including maintenance actions [do nothing]
- Arcs represent preconditions, adds and deletes
- We can only execute one real action at any step, but the data structure keeps track of all actions and states that are *possible*
- Add action in level A_i if ***all*** of its preconditions are present in level S_i
- Add literal in level S_i if it is the effect of ***some*** action in level A_{i-1} (*including no-ops*)
- Level S_0 has all of the literals from the initial state

ROCKET domain

- Literals:

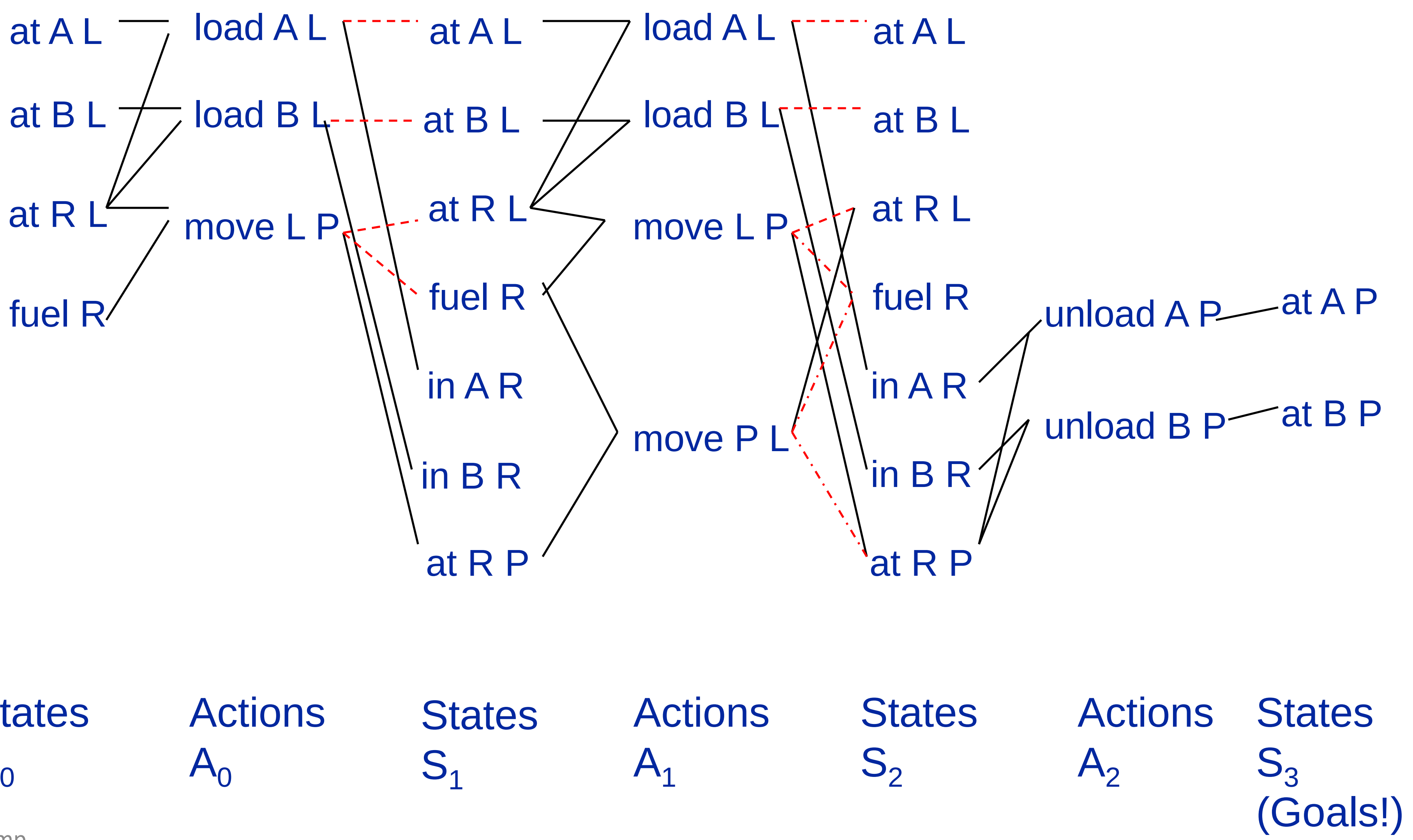
at X Y	X is at location Y
fuel R	rocket R has fuel
in X R	X is in rocket R

- Actions:

load X L	load X (onto R) at location L (X and R must be at L)
unload X L	unload X (from R) at location L (R must be at L)
move X Y	move rocket R from X to Y (R must be at L and have fuel)

- Graph representation:
  - Solid black lines: preconditions/effects
  - Dotted red lines: negated preconditions/effects

Example planning graph



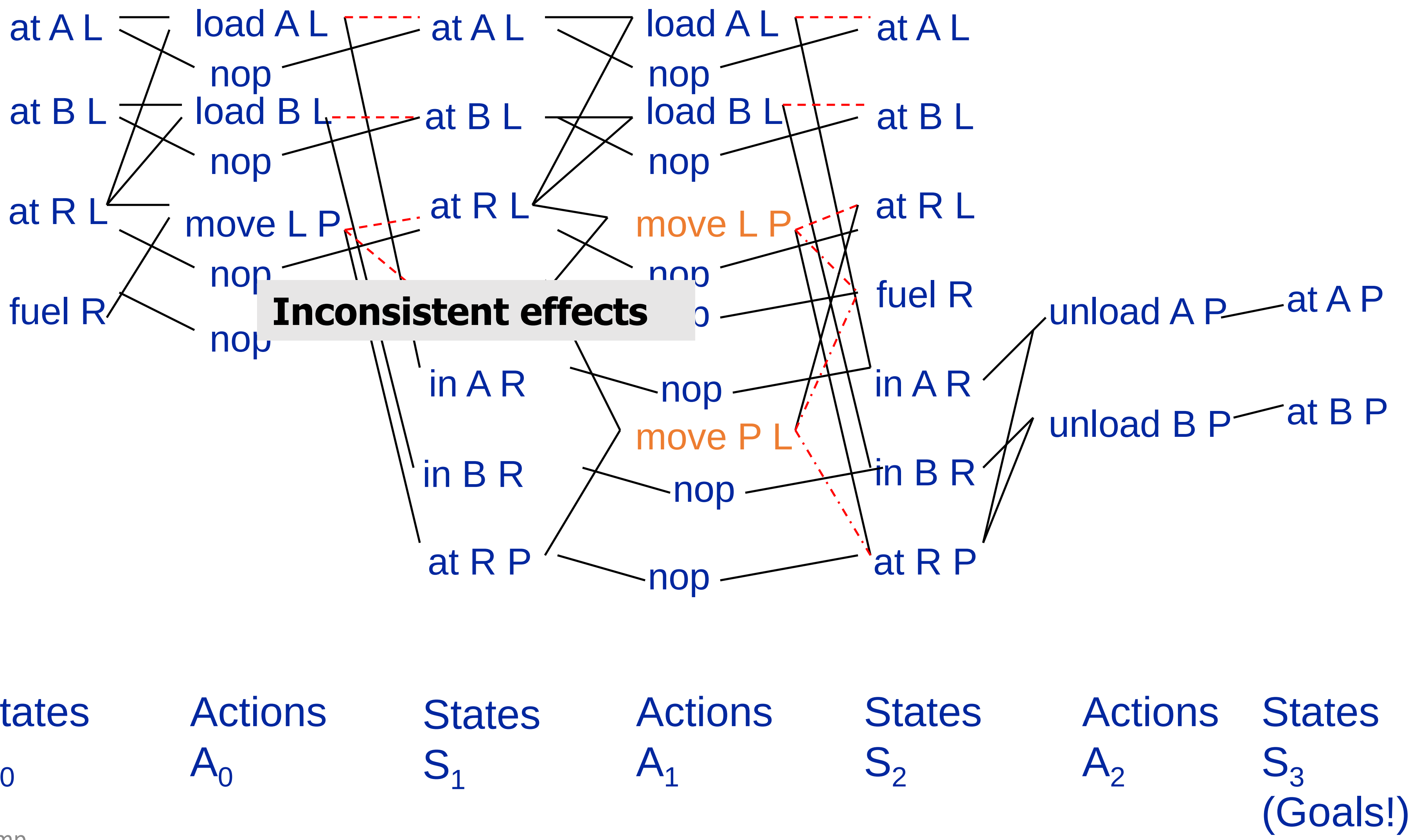
Valid plans

- A **valid** plan is a planning graph where:
  - Actions at the same level don’t interfere (delete each other’s preconditions or add effects)
  - Each action’s preconditions are true at that point in the plan
  - Goals are satisfied at the end of the plan

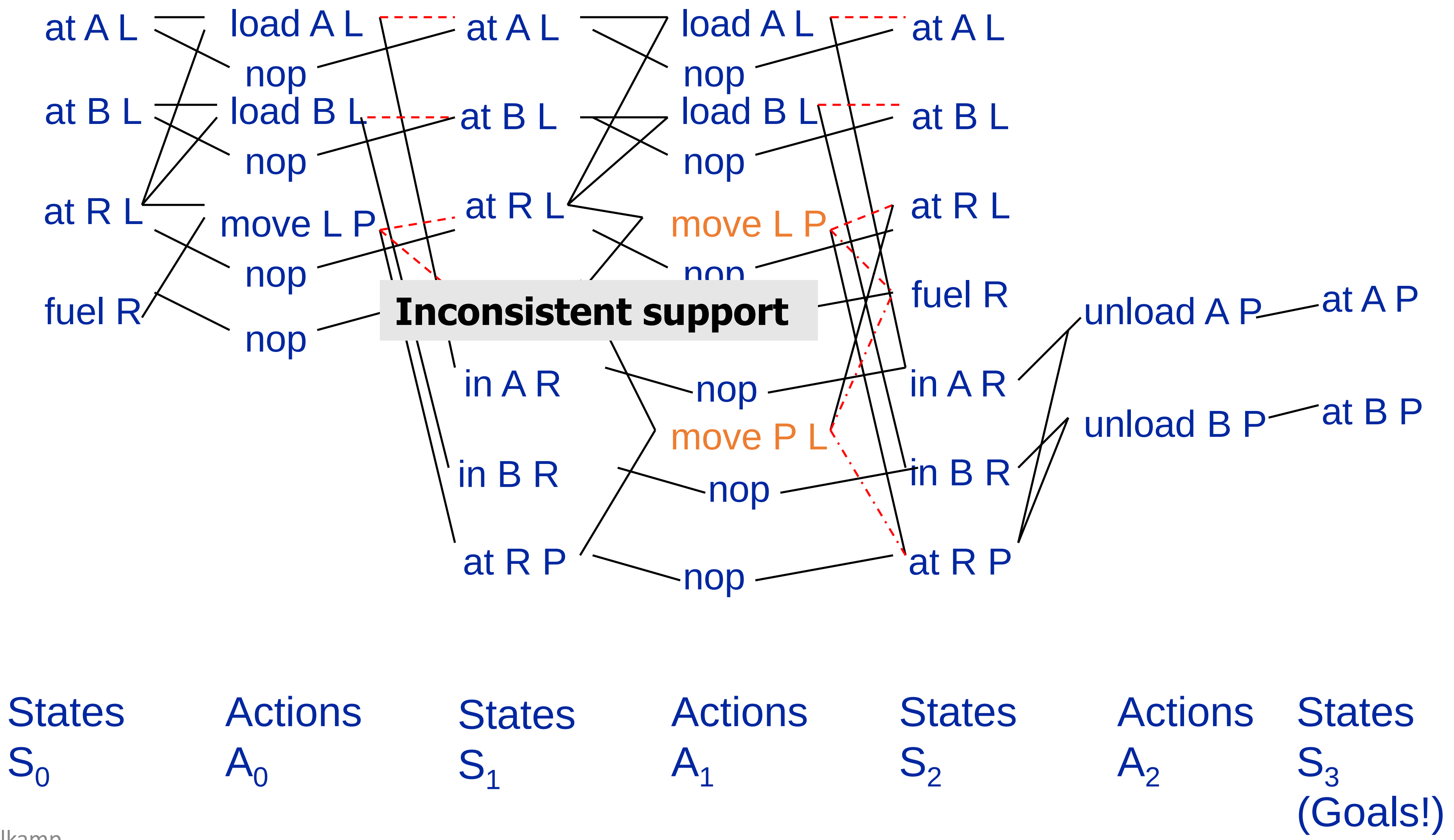
Exclusion relations (mutexes)

- Two actions (or literals) are **mutually exclusive ("mutex")** at step *i* if no valid plan could contain both.
- Can quickly find and mark *some* mutexes:
  - **Interference**: Two actions that interfere (the effect of one negates the precondition of another) are mutex
  - **Competing needs**: Two actions are mutex if any of their preconditions are mutex with each other
  - **Inconsistent support**: Two literals are mutex if all ways of creating them both are mutex

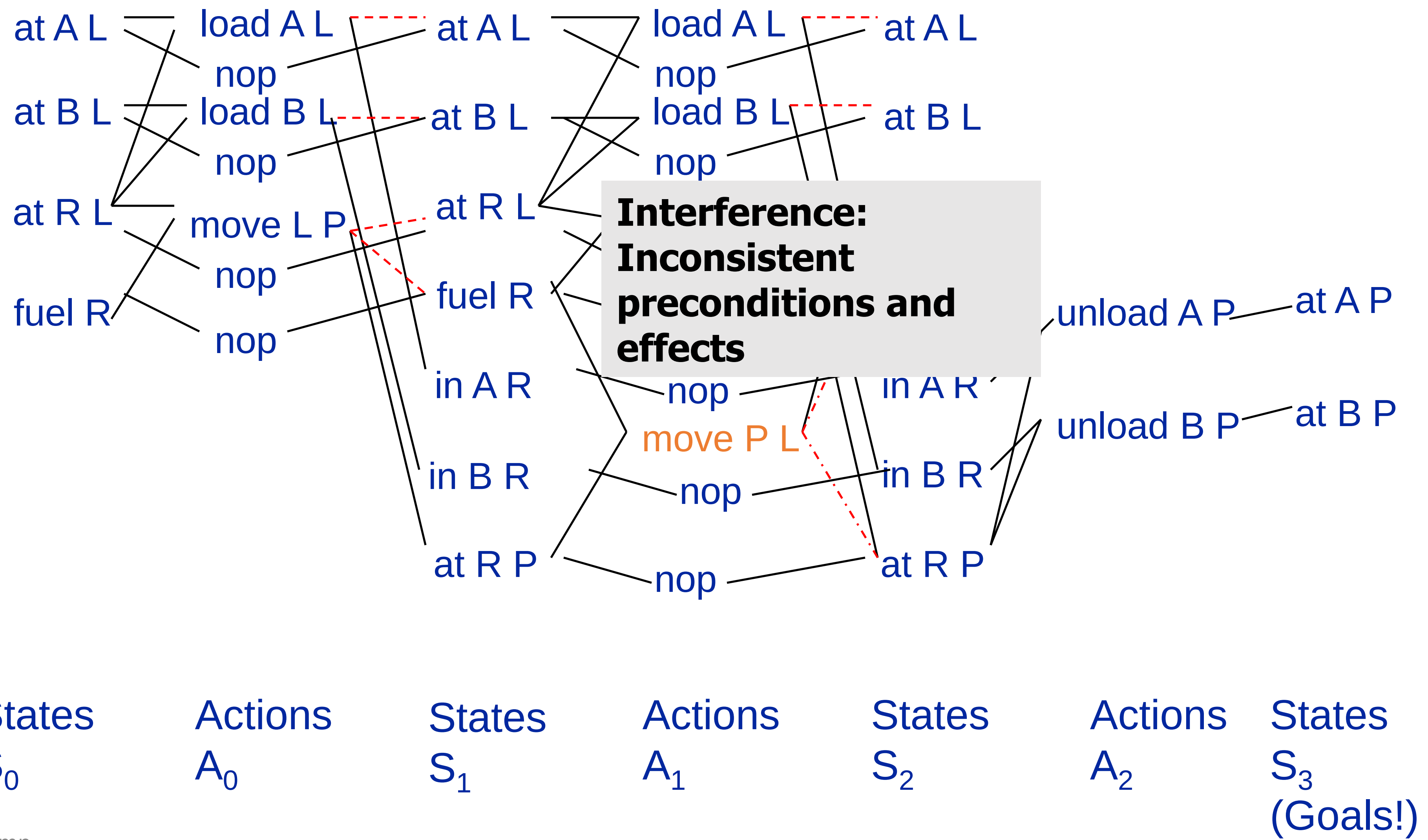
Example: Mutex constraints



Example: Mutex constraints

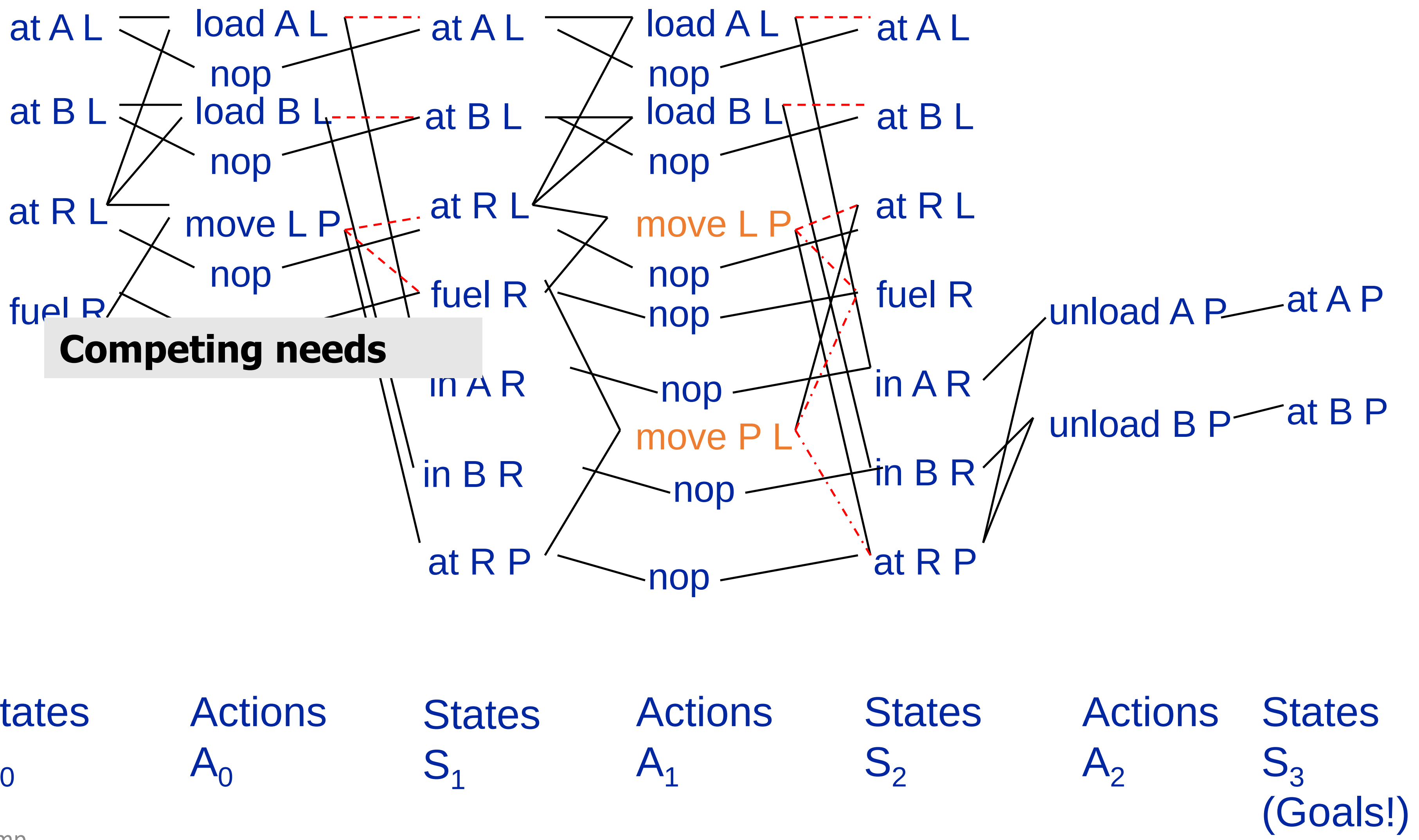


Example: Mutex constraints



Interference: Inconsistent preconditions and effects

Example: Mutex constraints



Extending the planning graph

- **Action level A_i:**
  - Include all instantiations of all actions (including maintains (no-ops)) that have all of their **preconditions satisfied** at level S_i, with no two being mutex
  - Mark as mutex all **action-maintain pairs** that are incompatible
  - Mark as mutex all **action-action pairs** that have competing needs
- **State level S_{i+1}:**
  - Generate all propositions that are the **effect of some action** at level A_i
  - Mark as mutex all pairs of propositions that can only be generated by **mutex action pairs**

Basic GraphPlan algorithm

- **Grow** the planning graph (PG) until all goals are reachable and none are pairwise mutex. (If PG levels off [reaches a steady state] first, fail)
- **Search** the PG for a **valid plan**
- If none found, **add a level** to the PG and try again

Creating the planning graph is usually fast

- Theorem:

  The size of the t-level planning graph and the time to create it are polynomial in:

  - t (number of levels),
  - n (number of objects),
  - m (number of operators), and
  - p (number of propositions in the initial state)

Searching for a plan

- Backward chain on the planning graph
- Complete all goals at one level before going back
- At level *i*, pick a non-mutex subset of actions that achieve the goals at level *i+1*. The preconditions of these actions become the goals at level *i*.
- Build the action subset by iterating over goals, choosing an action that has the goal as an effect. Use an action that was already selected if possible. Do forward checking on remaining goals.

The Big Three Planning Approaches

- Of the many planning approaches, three techniques stand out:
- 1 Graph/SAT Planning
- 2 Symbolic Search Planning
- 3 Heuristic State-Space Search

SAT Planning Encoding of STRIPS Planning Task (Variables)

Variables of SAT Encoding

- propositional variables v^i for all $v \in V$, $0 \leq i \leq T$ encode state after i steps of the plan
- propositional variables a^i for all $a \in A$, $1 \leq i \leq T$ encode actions applied in i-th step of the plan

SAT Planning of STRIPS Planning Task Clauses (for Initial and Goal State)

Clauses of SAT Encoding

- unit clauses encoding initial state:
  v^0 for all $v \in I$ and $\neg v_0$ for all $v \notin I$
- unit clauses encoding goal:
  v^T for all $v \in G$
- . . .

SAT Planning of STRIPS Planning Task Clauses (For Strips Actions)

Clauses of SAT Encoding

for all $1 \leq i \leq T$:

- subformulas encoding action preconditions:
  $a^i \rightarrow \bigwedge_{v \in pre(a)} v^{i-1}$
- subformulas encoding action conflicts:
  $\neg(a^i \wedge b^i)$ for all $a, b \in A$ with $a \neq b$,
  $(pre(a) \cup add(a)) \cap del(b) \neq \emptyset$
- subformulas encoding successor states:
  $v^i \leftrightarrow \left(\bigvee_{a \in A: v \in add(a)} a^i \vee (v^{i-1} \wedge \neg \bigvee_{a \in A: v \in del(a)} a^i) \right)$

Sat EnCoDing

- Formula describing the **initial state**: (let *E* be the set of possible facts in the planning problem)

  $\bigwedge\{e_0 \mid e \in I\} \wedge \bigwedge\{\neg e_0 \mid e \in E - I\}$

  Describes the complete initial state (both positive and negative fact)

  - E.g. $on(A,B,0) \wedge \neg on(B,A,0)$
- Formula describing the **goal**: (*G* is set of goal facts)

  $\bigwedge\{e_T \mid e \in G\}$

  says that the goal facts must be true in the final state at time step *T*
  - E.g. $on(B,A,T)$
- Is this enough?
  - Of course not. The formulas say nothing about actions.

Formulas in Φ

- For every action *a* and time step *i,* formula describing what fluents must be true if *a* were the *i*'th step of the plan:
  - $a_i \Rightarrow \bigwedge \{e_i \mid e \in \text{pre}(a)\}$, *a*'s preconditions must be true
  - $a_i \Rightarrow \bigwedge \{e_{i+1} \mid e \in \text{add}(a)\}$, a's ADD effects must be true in i+1
  - $a_i \Rightarrow \bigwedge \{\neg e_{i+1} \mid e \in \text{del}(a)\}$, a's DEL effects must be false in i+1
- *Complete exclusion* axiom:
  - For all actions *a* and *b* and time step *i*, formulas saying *a* and *b* can't occur at the same time

    $\neg a_i \vee \neg b_i$
  - this guarantees there can be only one action at a time

- Is this enough?
  - The formulas say nothing about what happens to facts if they are not effected by an action
  - This is known as the ***frame problem***

Frame Axioms

- *Frame axioms*:
  - Formulas describing what *doesn't* change between steps *i* and *i*+1
- Several are many ways to write these
  - Here I show a way that is good in practice
- **explanatory frame axioms**
  - One axiom for every possible fact *e at every* timestep *i*
  - Says that if *e* changes truth value between s_i and s_{i+1}, then the action at step *i* must be responsible:

$$\neg e_i \wedge e_{i+1} \Rightarrow \mathrm{V}\{a_i \mid \mathrm{e}\ in\ \mathrm{add}(a)\}$$

If *e* became true then some action must have added it

$$e_i \wedge \neg e_{i+1} \Rightarrow \mathrm{V}\{a_i \mid \mathrm{e}\ in\ \mathrm{del}(a)\}$$

If *e* became false then some action must have deleted it

Example

- Planning domain:
  - one robot r1
  - two adjacent locations l1, l2
  - one operator (move the robot)
- Encode (P,T) where $T = 1$

  - Initial state: {at(r1,l1)}
    Encoding: $at(r1,l1,0) \wedge \neg at(r1,l2,0)$

  - Goal: {at(r1,l2)}
    Encoding: $at(r1,l2,1)$

  - Action Schema: see next slide

Example (continued)

- Schema: move(r, l, l’)

  pre: at(r,l)

  add: at(r,l’)

  del: at(r,l)

  Encoding: (for actions move(r1,l1,l2) and move(r1,l2,l1) at time step 0)

  move(r1,l1,l2,0) $\Rightarrow$ at(r1,l1,0)

  move(r1,l1,l2,0) $\Rightarrow$ at(r1,l2,1)

  move(r1,l1,l2,0) $\Rightarrow$ $\neg$at(r1,l1,1)

  move(r1,l2,l1,0) $\Rightarrow$ at(r1,l2,0)

  move(r1,l2,l1,0) $\Rightarrow$ at(r1,l1,1)

  move(r1,l2,l1,0) $\Rightarrow$ $\neg$at(r1,l2,1)

Example (continued)

- Schema: move(r, l, l’)
  - pre: at(r,l)
  - add: at(r,l’)
  - del: at(r,l)

- Complete-exclusion axiom:
  - $\neg$move(r1,l1,l2,0) $\vee$ $\neg$move(r1,l2,l1,0)

- Explanatory frame axioms:
  - $\neg$at(r1,l1,0) $\wedge$ at(r1,l1,1) $\Rightarrow$ move(r1,l2,l1,0)
  - $\neg$at(r1,l2,0) $\wedge$ at(r1,l2,1) $\Rightarrow$ move(r1,l1,l2,0)
  - at(r1,l1,0) $\wedge$ $\neg$at(r1,l1,1) $\Rightarrow$ move(r1,l1,l2,0)
  - at(r1,l2,0) $\wedge$ $\neg$at(r1,l2,1) $\Rightarrow$ move(r1,l2,l1,0)

Complete Formula

$$
\begin{array}{l}
[\ at(r1,l1,0) \land \neg at(r1,l2,0)\ ] \land \\
at(r1,l2,1) \land \\
[\ move(r1,l1,l2,0) \Rightarrow at(r1,l1,0)\ ] \land \\
[\ move(r1,l1,l2,0) \Rightarrow at(r1,l2,1)\ ] \land \\
[\ move(r1,l1,l2,0) \Rightarrow \neg at(r1,l1,1)\ ] \land \\
[\ move(r1,l2,l1,0) \Rightarrow at(r1,l2,0)\ ] \land \\
[\ move(r1,l2,l1,0) \Rightarrow at(r1,l1,1)\ ] \land \\
[\ move(r1,l2,l1,0) \Rightarrow \neg at(r1,l2,1)\ ] \land \\
[\ \neg move(r1,l1,l2,0) \lor \neg move(r1,l2,l1,0)\ ] \land \\
[\ \neg at(r1,l1,0) \land at(r1,l1,1) \Rightarrow move(r1,l2,l1,0)\ ] \land \\
[\ \neg at(r1,l2,0) \land at(r1,l2,1) \Rightarrow move(r1,l1,l2,0)\ ] \land \\
[\ at(r1,l1,0) \land \neg at(r1,l1,1) \Rightarrow move(r1,l1,l2,0)\ ] \land \\
[\ at(r1,l2,0) \land \neg at(r1,l2,1) \Rightarrow move(r1,l2,l1,0)\ ]
\end{array}
$$

Convert to CNF and give to SAT solver.

Extracting a Plan

- Suppose we find an assignment of truth values that satisfies Φ.
  - This means *P* has a solution of length *n*
- For *I* = 0,…,*T-1*, there will be exactly one action *a* such that $a_i = true$
  - This is the *i*'th action of the plan.
- Example (from the previous slides):
  - Φ can be satisfied with move(r1,l1,l2,0) = *true*
  - Thus $\langle$move(r1,l1,l2,0)$\rangle$ is a solution for (*P*,0)
    - It's the only solution - no other way to satisfy Φ

Supporting Layered Plans (as in Graphplan) "Blackbox"

- *Complete exclusion* axiom:
  - For **all** actions *a* and *b* and time steps *i* include the formula $\neg a_i \vee \neg b_i$
  - this guaranteed that there could be only one action at a time
- *Partial exclusion axiom*:
  - For any pair of incompatible actions *a* and *b* and each time step *i* include the formula $\neg a_i \vee \neg b_i$
  - This encoding will allowed for more than one action to be taken at a time step resulting in layered plans
  - This is advantageous because fewer time steps are required (i.e. shorter formulas)

The Big Three Planning Approaches

- Of the many planning approaches, three techniques stand out:
- 1 Graph/SAT Planning
- 2 Symbolic Search Planning
- 3 Heuristic State-Space Search

Symbolic Search Planning: BASIC Idea

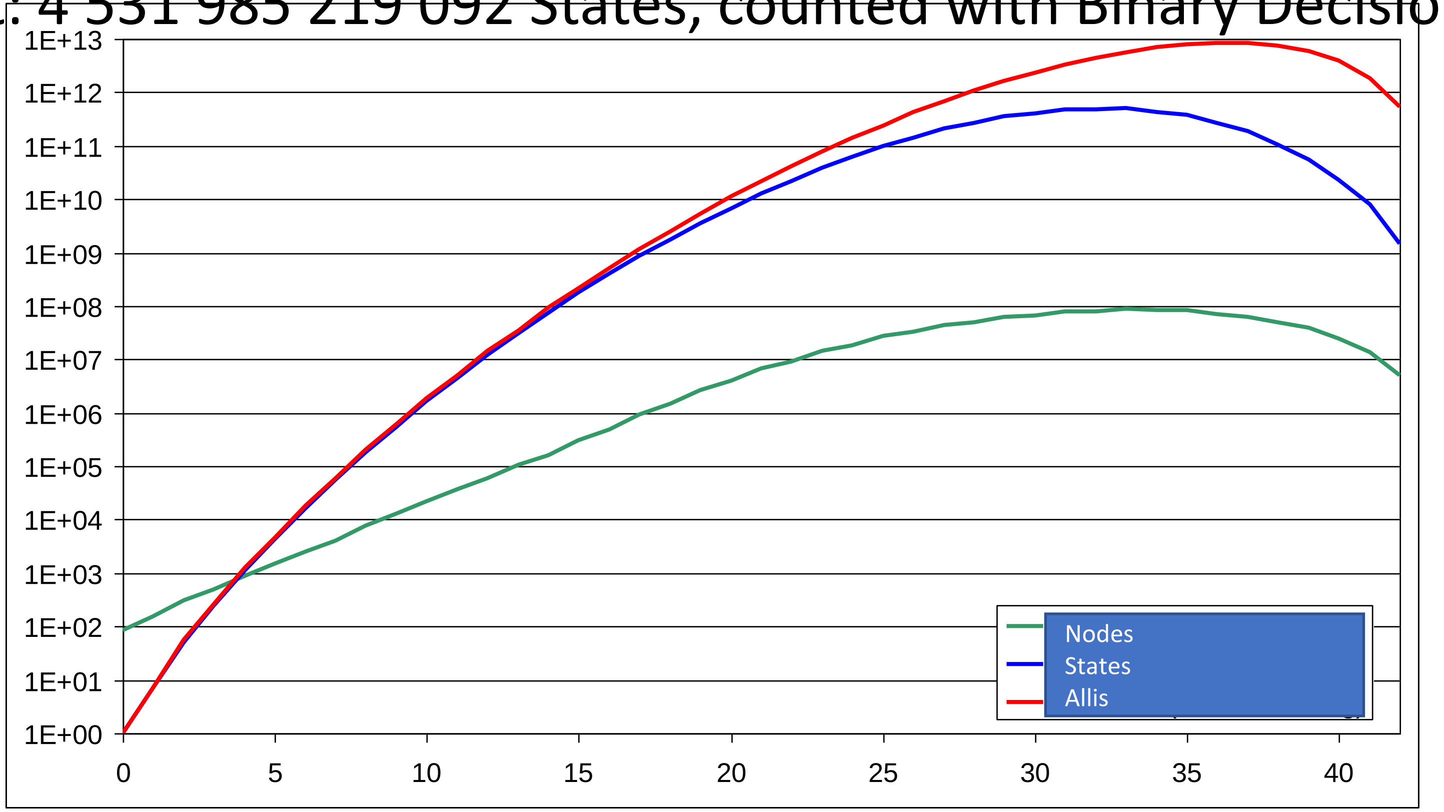
- ➢search processes sets of states at a time
- ➢operators, goal states, state sets reachable with a given cost

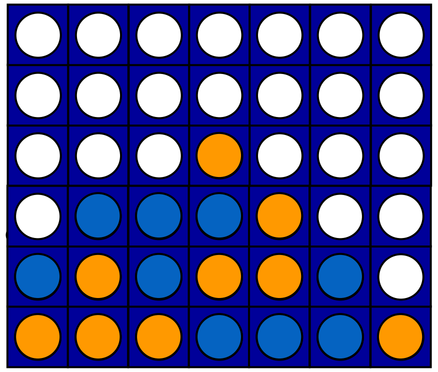
  etc. represented by binary decision diagrams (BDDs) (or related data structures)

- **hope**: exponentially large state sets can be represented as polynomially sized BDDs, which can be efficiently processed
- → perform symbolic search on these set representations

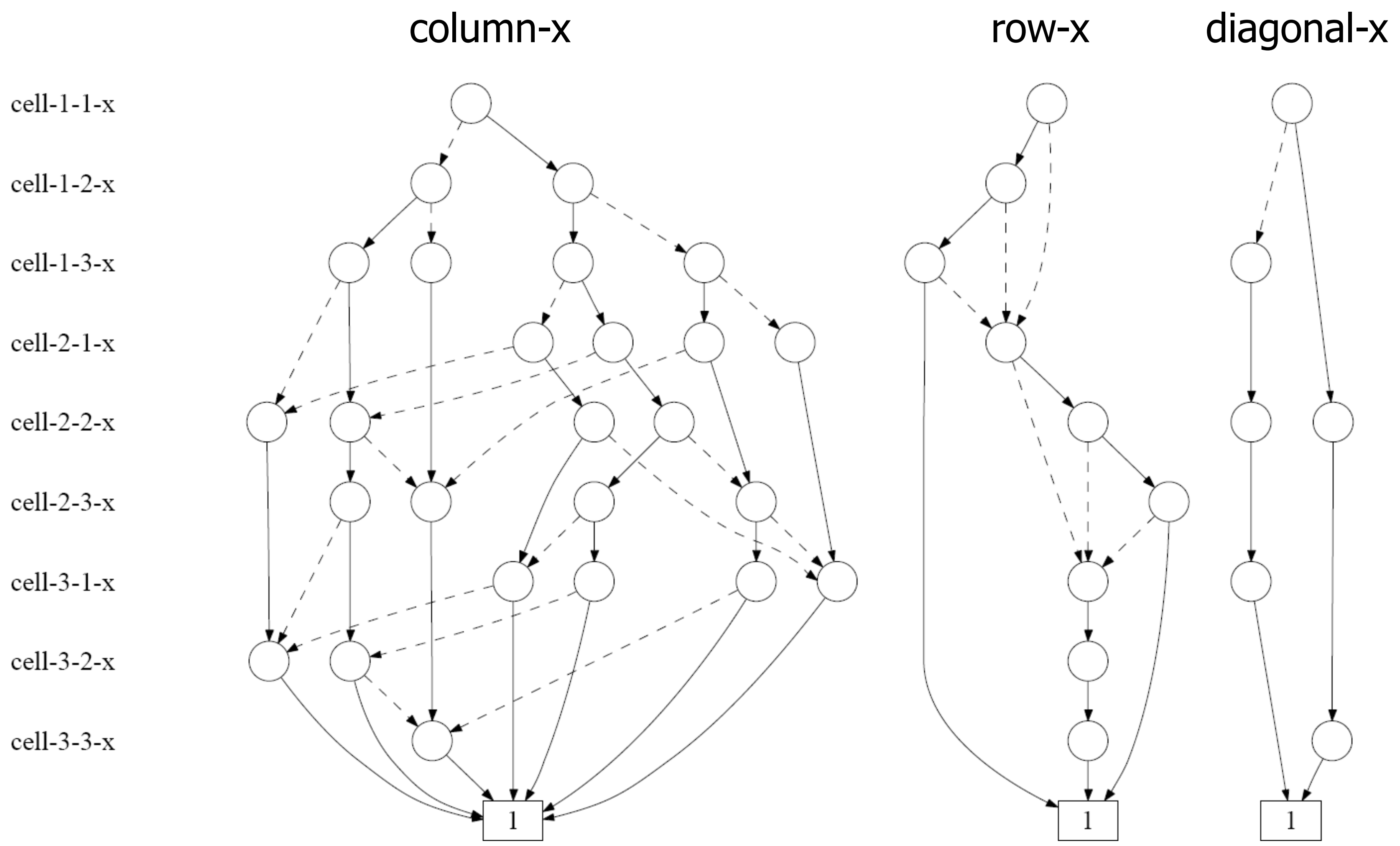
Representation Gain: Connect Four

- 1988 solved by Victor Allis and James D. Allen
  - Prediction Allis: 70 728 639 995 483 States
  - But: 4 531 985 219 092 States, counted with Binary Decision Di



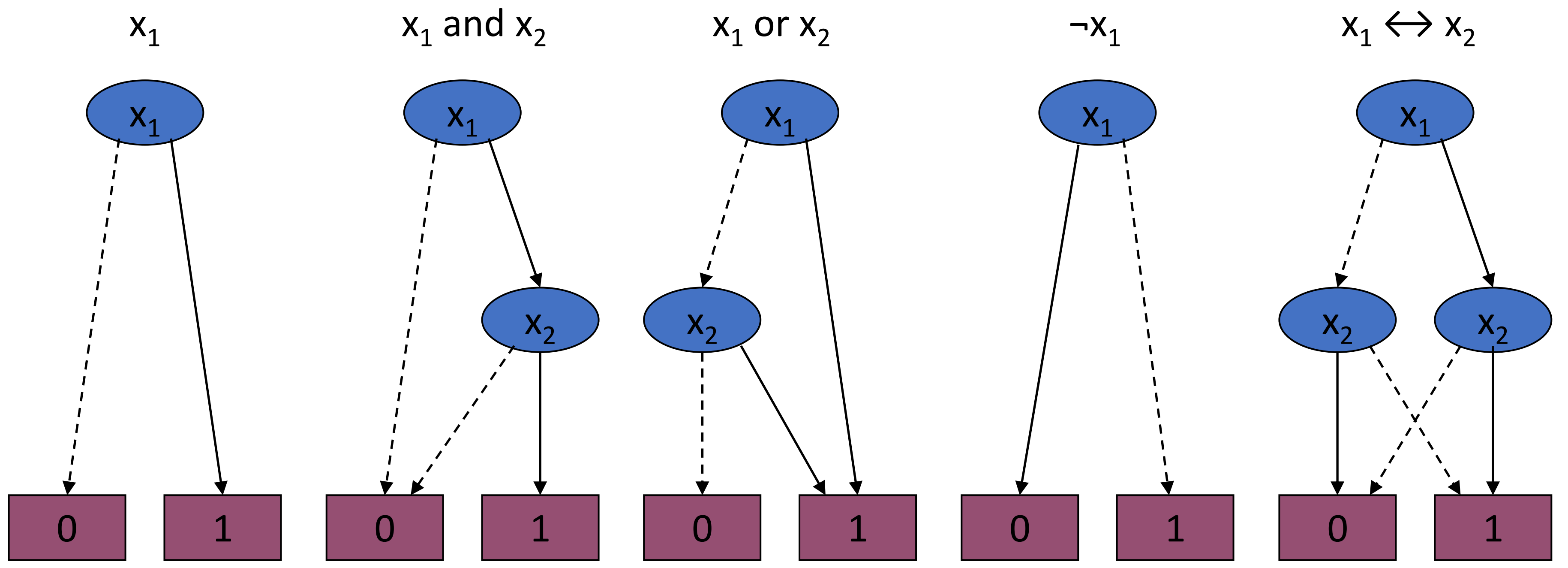


BDDs (Example)



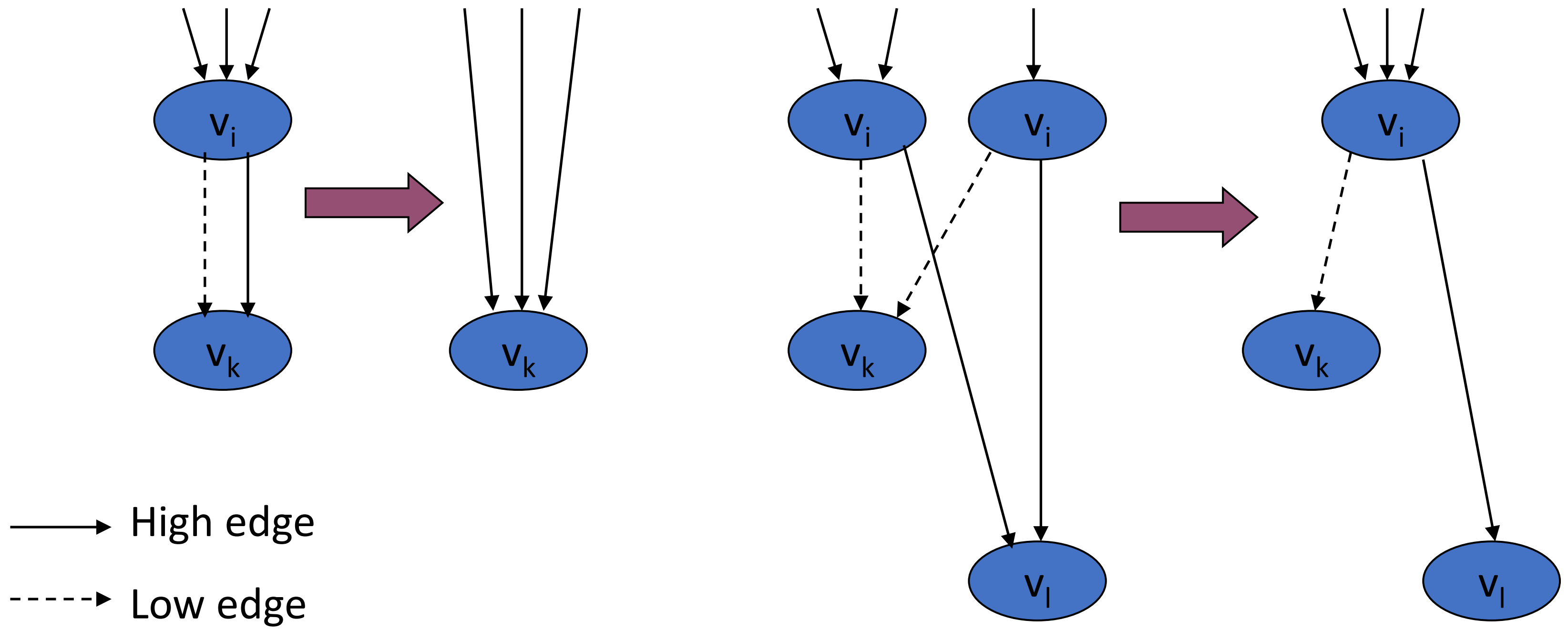
BDDs

BDDs for simple Boolean functions



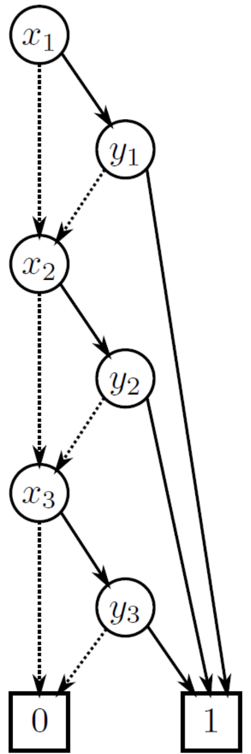
BDD Reduction

- Fixed variable ordering, 2 reduction rules



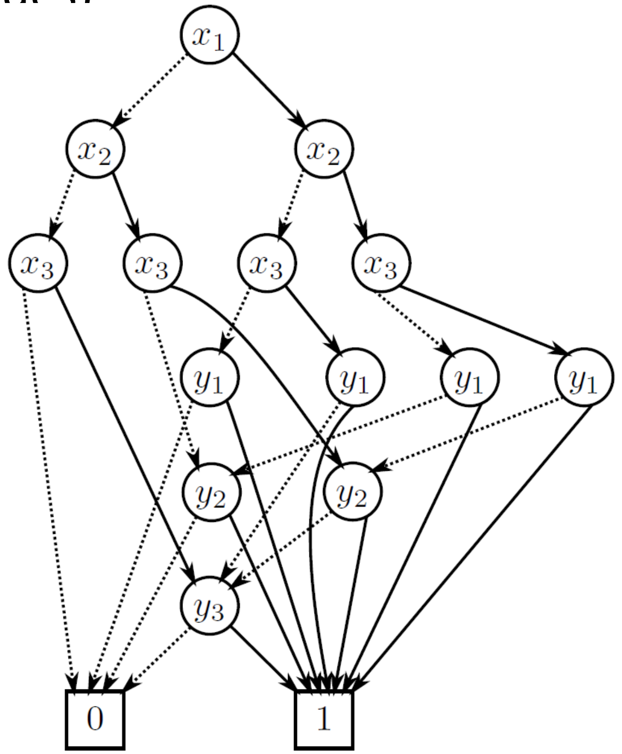
Variable Ordering

- Example: Disjunctive Quadratic Form (DQF)
  - $DQF_n(x_1, \dots, x_n, y_1, \dots, y_n) := (x_1 \text{ and } y_1) \text{ or } (x_2 \text{ and } y_2) \text{ or } \dots \text{ or } (x_n \text{ and } y_n)$
  - Ordering $\pi = (x_1, y_1, x_2, y_2, \dots, x_n, y_n)$:
    - Linear size (2n + 2 nodes)



Variable ordering

- Example: Disjunctive Quadratic Form (DQF)
  - $DQF_n(x_1, \ldots, x_n, y_1, \ldots, y_n) := (x_1 \text{ and } y_1) \text{ or } (x_2 \text{ and } y_1) \text{ or } \ldots \text{ or } (x_n \text{ and } y_n)$
  - Ordering $\pi = (x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$:
    - Exponential size (2^{n+1} nodes)

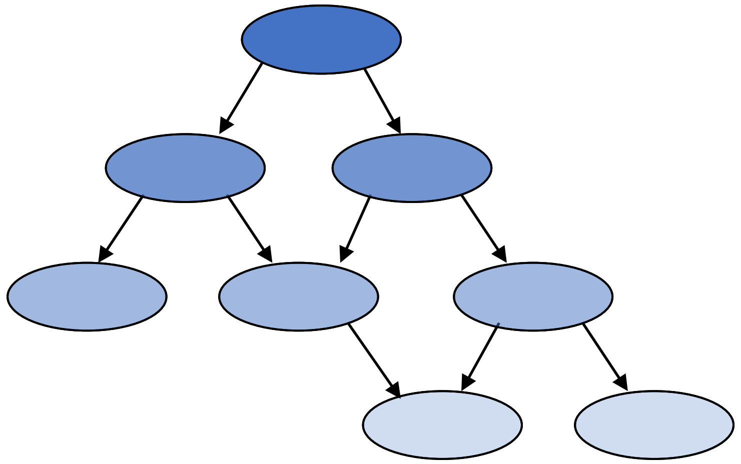


Transition relation

- Two variable sets S and S'
  - BDD for Action a_i = (pre, eff)
    - $trans_i(S, S') := pre_{BDD}(S) \wedge eff_{BDD}(S') \wedge frame_{BDD}(S, S')$
  - $frame_{BDD}$: models frame
    - all what doen't change (has to be computed – all what is neither add or del effect remains unchanged)
    - If v_1 and v_2 unchanged:
      - $frame_{BDD}(S, S') := (v_1(S) \leftrightarrow v_1(S')) \wedge (v_2(S) \leftrightarrow v_2(S'))$

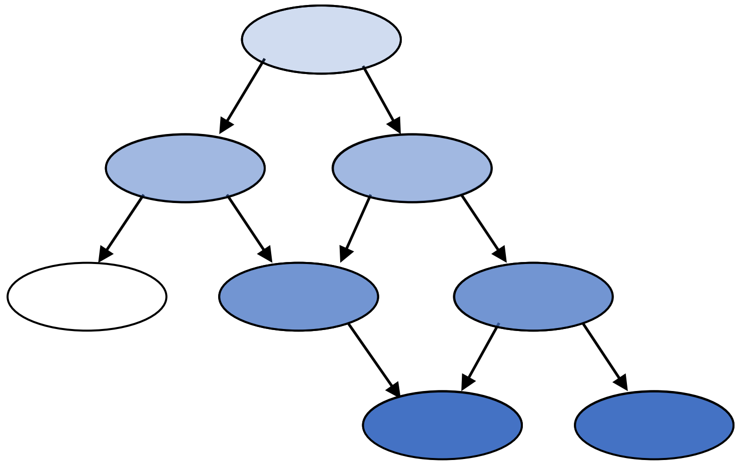
Successor Computation

- Find all successors of set with image function
  - image(current) := (∃S: trans(S, S') ^ current(S)) [S' → S]
  - current: given state set
  - [S' → S]: moving successor set variables to current ones
    - Needed to continue the search



Predecessor Computation

- Finding all predecessors similar (only exchange of variable sets)
    - pre-image(current) := (∃S’: trans(S, S’) ^ current(S’)) [S → S’]



Partitioned Successor Computation
Relational Product / Apply

$$\begin{aligned}
image\,(current) &= (\exists S.trans\,(S,S') \wedge current\,(S))\,[S' \to S] \\
&\equiv \left(\exists S.\left(\bigvee_{a\in\mathcal{A}} trans_a\,(S,S')\right) \wedge current\,(S)\right)[S' \to S] \\
&\equiv \left(\exists S.\bigvee_{a\in\mathcal{A}} (trans_a\,(S,S') \wedge current\,(S))\right)[S' \to S] \\
&\equiv \bigvee_{a\in\mathcal{A}} (\exists S.trans_a\,(S,S') \wedge current\,(S))\,[S' \to S]
\end{aligned}$$

Finite Domain Variable Encoding

Planning state = set of finite domain (SAS+) variables

- Finite domain variables can be encoded in binary
  - for variable with domain of size n we need ⌈log(n)⌉ binary ones
    - for n=7:
      - Value 0 equals 000
      - Value 1 equals 001
      - Value 2 equals 010
      - etc.
  - so that state is a conjunction of variables.

Symbolic Planning: Symbolic BFS

Progression Breadth-first Search

```
def bfs-progression($V$, $I$, $O$, $G$):
    $goal\_states := conjunction(G)$
    $reached_0 := \{I\}$
    $i := 0$
    loop:
        if $reached_i \cap goal\_states \neq \emptyset$:
            return solution found
        $reached_{i+1} := reached_i \cup apply(reached_i, O)$
        if $reached_{i+1} = reached_i$:
            return no solution exists
        $i := i + 1$
```

SymboliC Dijkstra

```
• $open_0 \leftarrow \mathcal{I}$, closed $\leftarrow \bot$, $g \leftarrow 0$
• repeat
    ▪ if ($open_g \wedge \mathcal{G} \neq \bot$) STOP
    ▪ $open_g \leftarrow open_g \wedge !closed$
    ▪ für $c \leftarrow 1, \dots, C$
        • $open_{g+c} \leftarrow open_{g+c} \vee image_c(open_g)$
    ▪ closed $\leftarrow$ closed $\vee open_g$
    ▪ $g \leftarrow g + 1$
```

Symbolic A* (BDDA*)