

# 15-780: Grad AI

## Lecture 13: Duality; Planning

---

*Geoff Gordon (this lecture)*

*Tuomas Sandholm*

*TAs Sam Ganzfried, Byron Boots*



# Review

# Branch & bound (& cut)



- *Worked examples*
- *Demonstrated how to simulate resolution*
  - *and therefore DPLL+CL*

# MILP examples



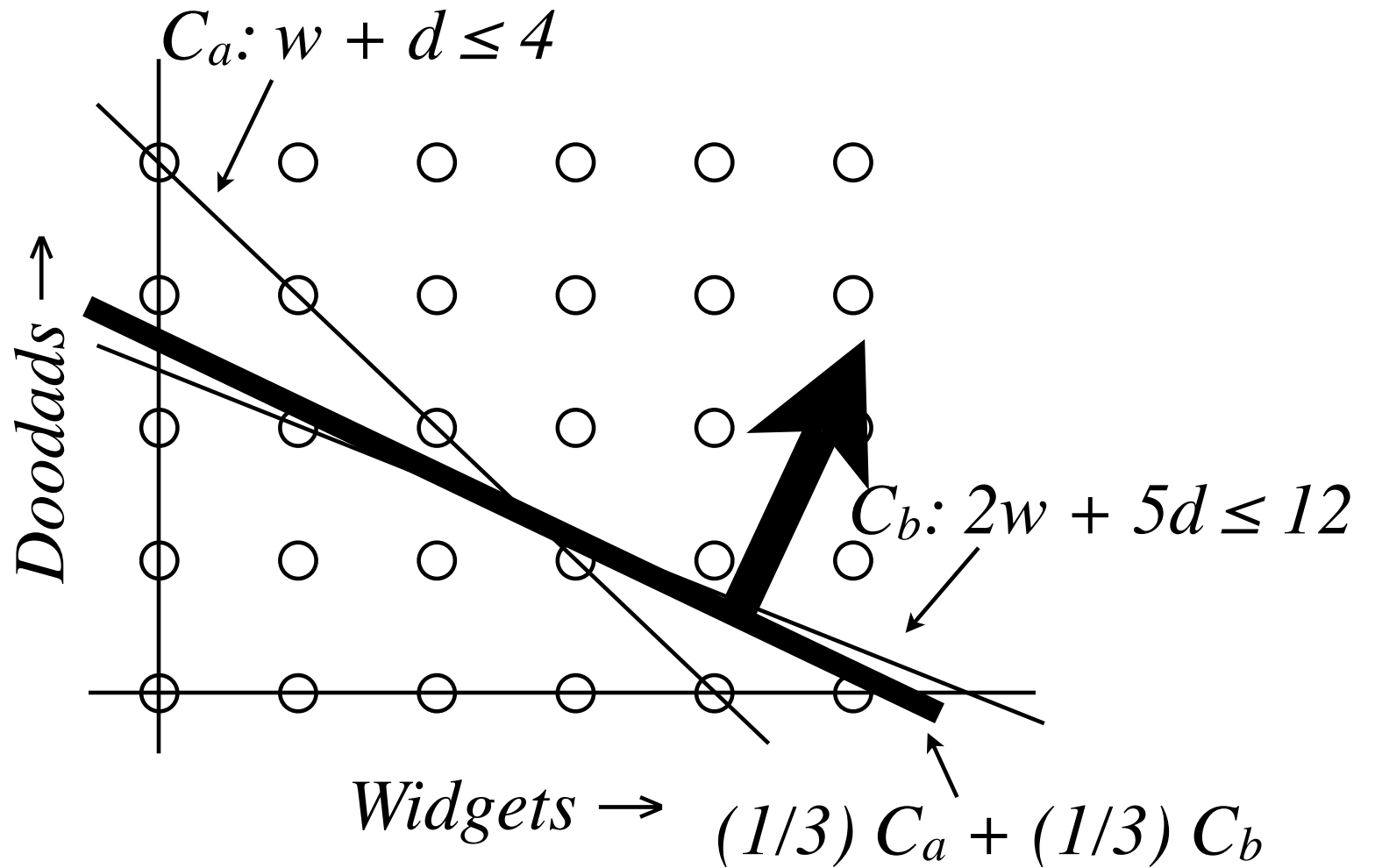
- *Path planning*
  - *not NP-complete; 0 integrality gap*
- *Planetary exploration planning*

# Duality w/ inequalities

---

- *Take a linear combination of constraints to bound objective*
- $(a + 2b)w + (a + 5b)d \leq 4a + 12b$
- $\text{profit} = 1w + 2d$
- *So, if  $1 \leq (a + 2b)$  and  $2 \leq (a + 5b)$ , we know that  $\text{profit} \leq 4a + 12b$*

# Duality picture



# Use of duality

---

- *Any feasible solution to dual yields upper bound (compared with only optimal solution to primal)*
- *Dual sometimes easier to work with*

# Dual dual

---

- *Take the dual of an LP twice, get the original LP back (called **primal**)*
- *Many LP solvers will give you both primal and dual solutions at the same time for no extra cost*



# Duality w/ equality

# Equality example



- *minimize  $y$  subject to*
- $x + y = 1$
- $2y - z = 1$
- $x, y, z \geq 0$

# Equality example

---

- *Want to prove bound  $y \geq \dots$*

- *Look at 2nd constraint:*

$$2y - z = 1 \quad \Rightarrow$$

$$y - z/2 = 1/2$$

- *Since  $z \geq 0$ , dropping  $-z/2$  can only increase LHS  $\Rightarrow$*

- $y \geq 1/2$

# Duality w/ equalities

---

- *In general, could start from any linear combination of equality constraints*
  - *no need to restrict to +ve combination*
- $a(x + y - 1) + b(2y - z - 1) = 0$
- $ax + (a + 2b)y - bz = a + b$

# Duality w/ equalities

---

- $a x + (a + 2b) y - b z = a + b$
- *As long as coefficients on LHS  $\leq (0, 1, 0)$ ,*
  - *objective =  $0 x + 1 y + 0 z \geq a + b$*
- *So, maximize  $a + b$  subject to*
  - $a \leq 0$
  - $a + 2b \leq 1$
  - $-b \leq 0$



# Duality recipes

# Recipe for inequalities

---

◦ *If we have an LP in matrix form,*

*maximize  $c'x$  subject to*

$$Ax \leq b$$

$$x \geq 0$$

◦ *Its dual is a similar-looking LP:*

*minimize  $b'y$  subject to*

$$A'y \geq c$$

$$y \geq 0$$

*$Ax \leq b$  means every component of  $Ax$  is  $\leq$  corresponding component of  $b$*

# Recipe with $\leq$ and $=$

- *If we have an LP with equalities,*

*maximize  $c'x$  s.t.*

$$Ax \leq b$$

$$Ex = f$$

$$x \geq 0$$

- *Its dual has some unrestricted variables:*

*minimize  $b'y + f'z$  s.t.*

$$A'y + E'z \geq c$$

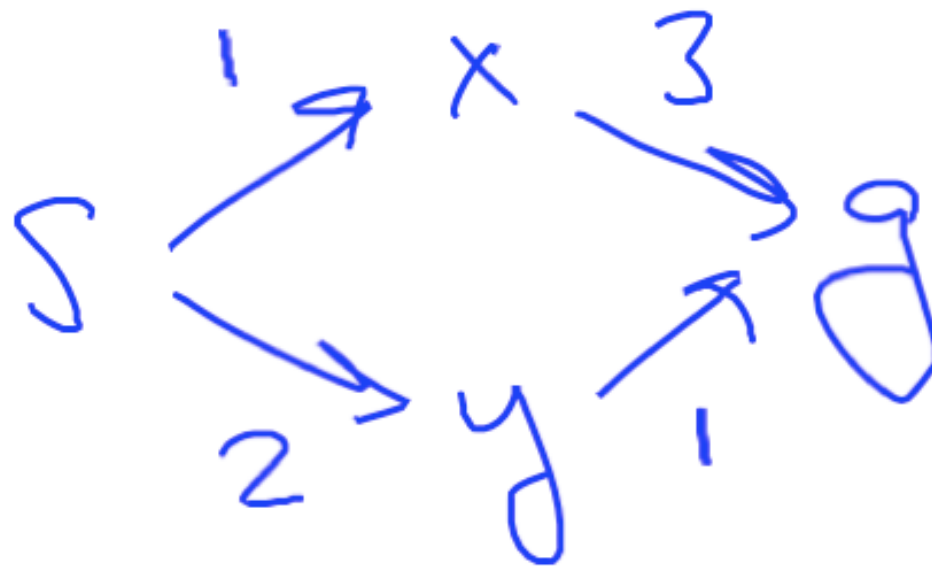
$$y \geq 0$$

*$z$  unrestricted*



# Duality example

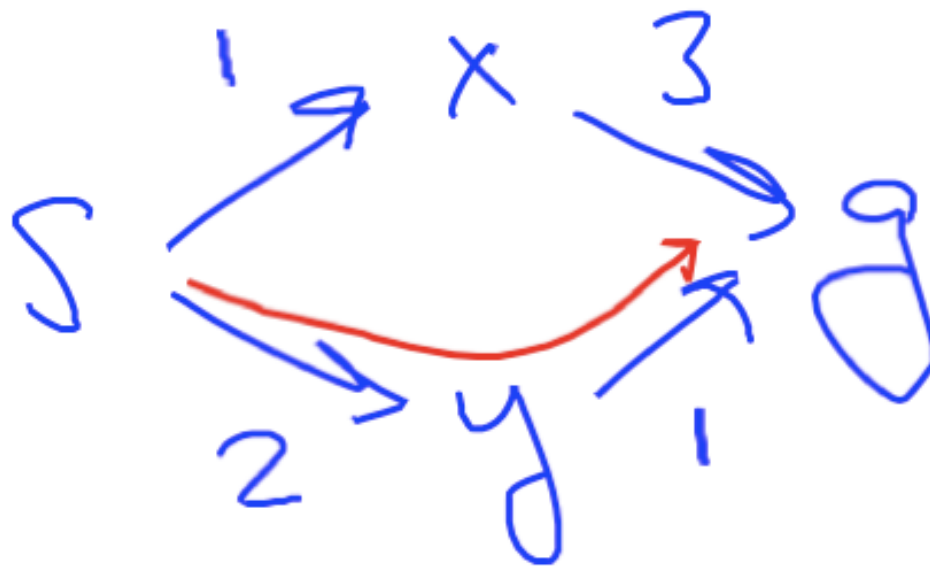
# Path planning LP



- Find the min-cost path: variables

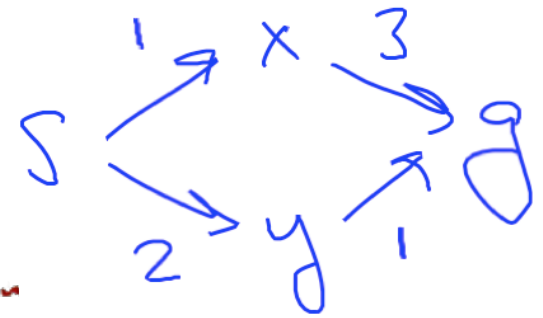
$$P_{sx}, P_{sy}, P_{xg}, P_{yg} \geq 0$$

# Optimal solution



$$p_{sy} = p_{yg} = 1, \quad p_{sx} = p_{xg} = 0, \quad \text{cost } 3$$

# Path planning LP



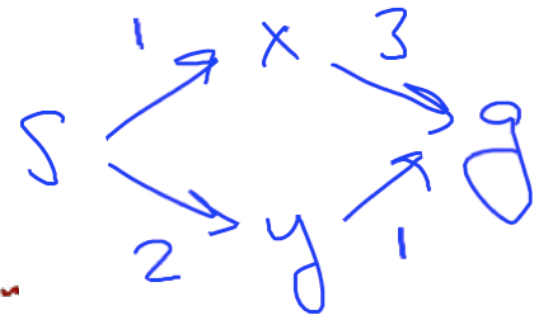
$$\text{Min } (1 \ 3 \ 2 \ 1) P$$

st

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & -1 & 0 & -1 \end{pmatrix} P = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$P \geq 0$$

# Path planning LP



$$\text{Min } (1 \ 3 \ 2 \ 1) p$$

st

$$\begin{matrix} \lambda_s \\ \lambda_x \\ \lambda_y \\ \lambda_g \end{matrix} \begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 0 & -1 \end{pmatrix} p = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$p \geq 0$$

# Deriving dual

$$\min c^T p \quad \text{s.t.}$$

$$A p = b \quad p \geq 0$$

---

$$\lambda^T A p = \lambda^T b$$

← linear combo  
of constr

←  $p \geq 0$

$$\lambda^T A \leq c^T \Rightarrow \lambda^T A p \leq c^T p$$

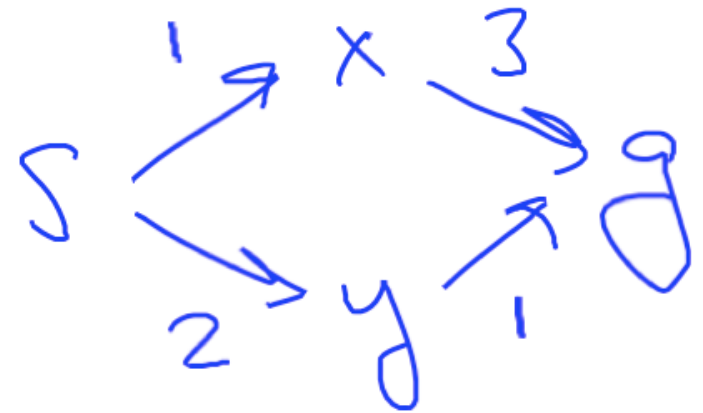
$$\Rightarrow \lambda^T b \leq c^T p \quad \leftarrow \text{bound}$$

$$\max \lambda^T b \quad \text{s.t.}$$

$$\lambda^T A \leq c^T \Leftrightarrow A^T \lambda \leq c$$

# Dual

$$\begin{aligned} \max \quad & \lambda_s - \lambda_g \\ \text{st} \quad & \lambda_s - \lambda_x \leq 1 \\ & \lambda_x - \lambda_g \leq 3 \\ & \lambda_s - \lambda_g \leq 2 \\ & \lambda_x - \lambda_g \leq 1 \end{aligned}$$



# Optimal dual solution

$$\begin{array}{l}
 \max \quad \lambda_s - \lambda_g \\
 \text{st} \quad \lambda_s - \lambda_x \leq 1 \\
 \quad \lambda_x - \lambda_g \leq 3 \\
 \quad \lambda_s - \lambda_g \leq 2 \\
 \quad \lambda_g - \lambda_x \leq 1
 \end{array}$$

*Any solution which adds a constant to all  $\lambda$ s also works;  $\lambda_x = 2$  also works*



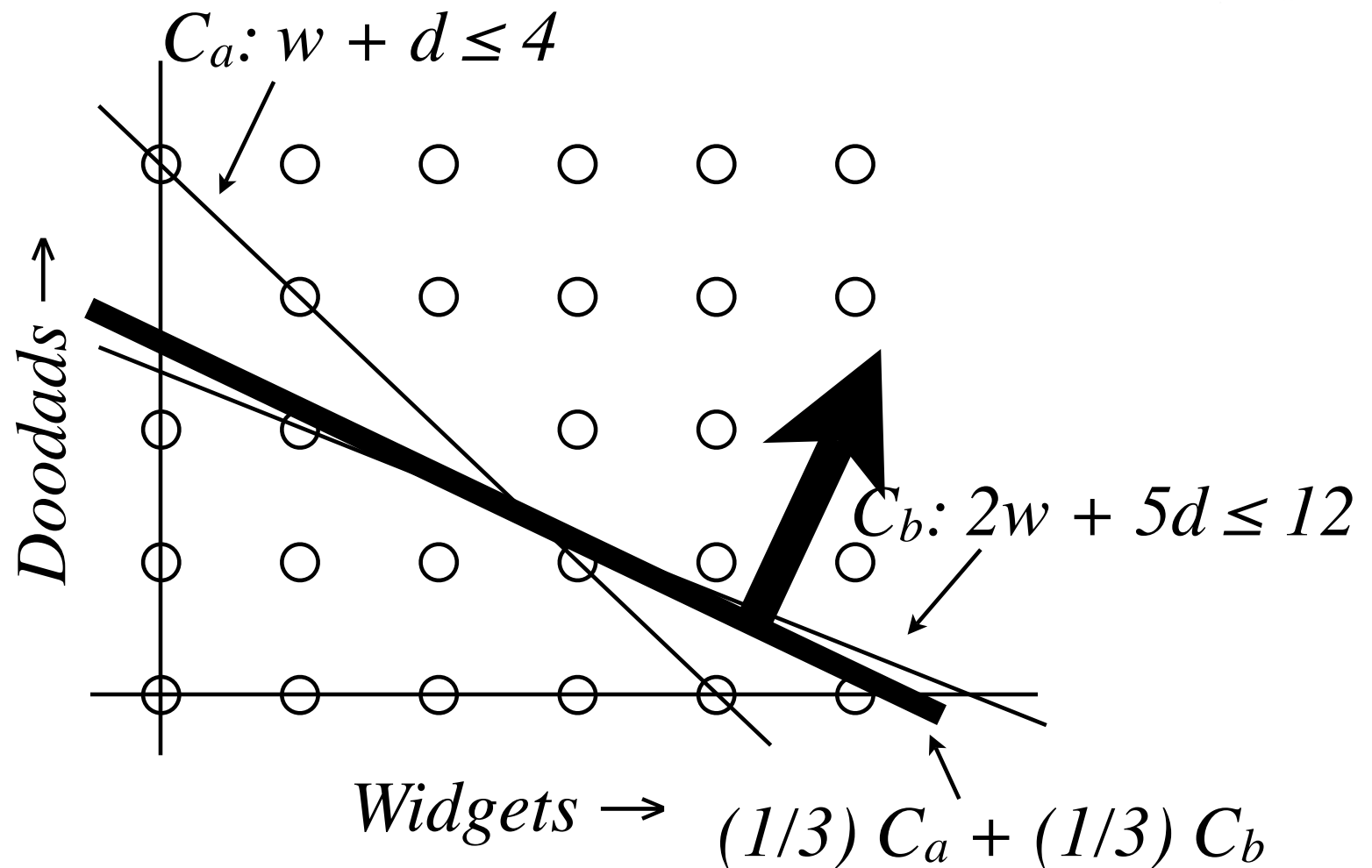
# Interpreting the dual

# Interpreting the dual variables

---

- *The primal variable variables in the factory LP were how many widgets and doodads to produce*
- *We interpreted dual variables as multipliers for primal constraints*

# Dual variables as multipliers



# Dual variables as prices



- *“Multiplier” interpretation doesn’t give much intuition*
- *It is often possible to interpret dual variables as **prices** for primal constraints*

# Dual variables as prices

---

- *Suppose someone offered us a quantity  $\varepsilon$  of wood, loosening constraint to*

$$w + d \leq 4 + \varepsilon$$

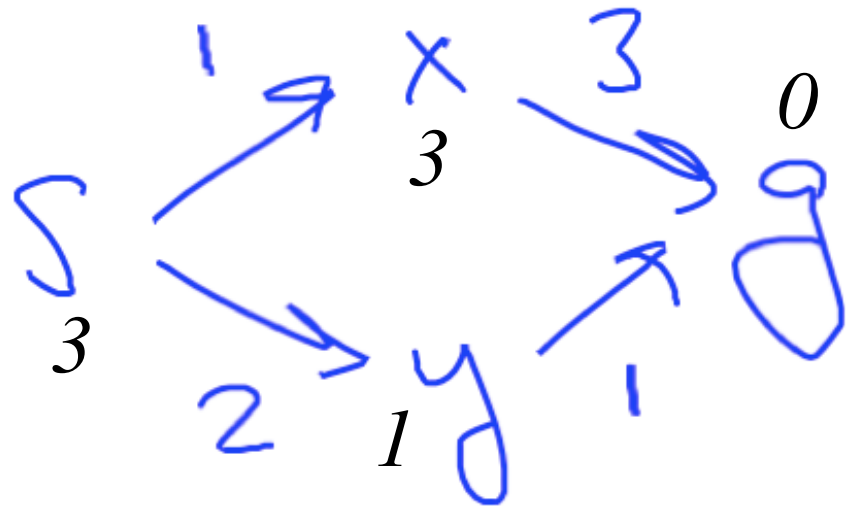
- *How much should we be willing to pay for this wood?*

# Dual variables as prices

---

- *RHS in primal is objective in dual*
- *So, dual constraints stay same, previous solution  $a = b = 1/3$  still dual feasible*
  - *still optimal if  $\varepsilon$  small enough*
- *Bound changes to  $(4 + \varepsilon) a + 12 b$ , difference of  $\varepsilon * 1/3$*
- *So we should pay up to \$1/3 per unit of wood (in small quantities)*

# Price example: path planning



- *Dual variables are prices on nodes: how much does it cost to start there?*
- *Dual constraints are local price constraints: edge  $xg$  (cost 3) means that node  $x$  can't cost more than  $3 + \text{price of node } g$*



# Planning

# Time



- *Recall fluents*
- *For KBs that evolve, add extra argument to each predicate saying when it was true*
  - *at(Robot, Wean5409)*
  - *at(Robot, Wean5409, 17)*

# Operators

- *Given a representation like this, can define operators that change state*
- *E.g., given*
  - *at(Robot, Wean5409, 17)*
  - *moves(Robot, Wean5409, corridor, 17)*
- *results might be*
  - *at(Robot, corridor, 18)*
  - $\neg$ *at(Robot, Wean5409, 18)*

# Goals



- *Want our robot to, e.g., get sandwich*
- *Search for proof of  $\text{has}(\text{Geoff}, \text{Sandwich}, t)$*
- *Try to analyze proof tree to find sequence of operators that make goal true*

# Complications

---

- *This strategy yields lots of complications*
  - *need axioms describing natural numbers (for time)*
  - *frame or successor-state axioms (facts don't change unless operator does it)*
  - *unique names, exactly one action per step, generalization of answer literal...*
- *Result can be slow inference*

# Planning



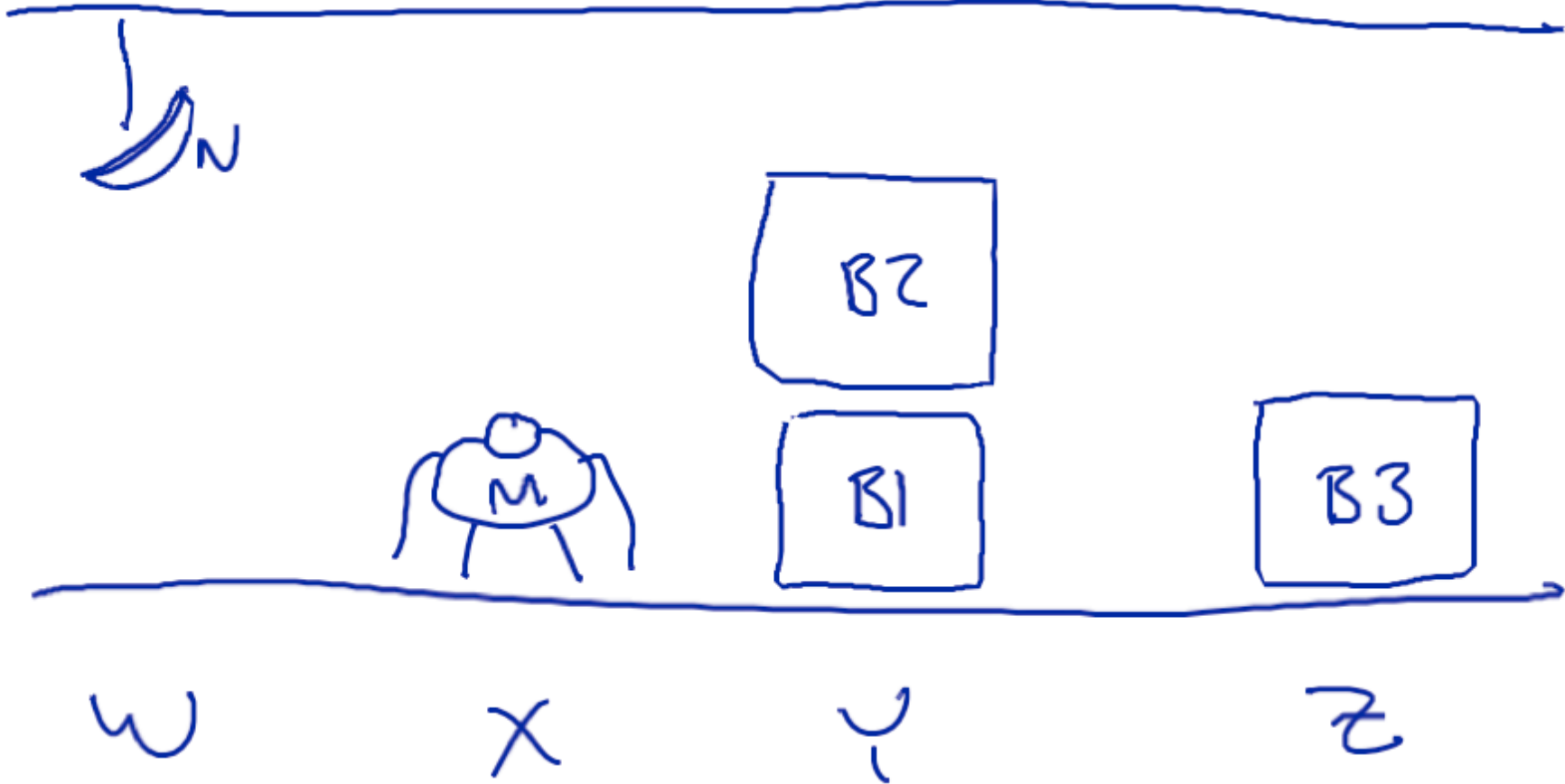
- *Alternate solution: define a subset of FOL especially for planning*
- *E.g., STRIPS language*
  - *no functions, limited quantification, ...*
- *STanford Research Institute Problem Solver*

# STRIPS

---

- *State of world at each time =  
{ propositions }*
- *Each proposition is ground literal*
- *For brevity, list only true literals*
- *Time is implicit*

# STRIPS state example



# STRIPS state example

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, X)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *height(M, Low)*
- *height(N, High)*

# STRIPS operators

---

- *Operator = { preconditions }, { effects }*
- *If preconditions are true at time  $t$ ,*
  - *can apply operator at time  $t$*
  - *effects will be true at time  $t+1$*
  - *rest of state unaffected*
- *Basic STRIPS: one operator per step*

# Quantification in operators

---

- *Preconditions of operator may contain variables (implicit  $\forall$ )*
- *Operator can apply if preconditions unify with state  $t$  (using binding  $X$ )*
- *state  $t+1$  has  $e / X$  for each  $e$  in effects*

# Operator example

---

- $Eat(target, p, l)$ 
  - **pre**:  $hungry(M), food(target), at(M, p), at(target, p), level(M, l), level(target, l)$
  - **eff**:  $\neg hungry(M), full(M), \neg at(target, p), \neg level(target, l)$

# Operator example

---

- *Move(from, to)*
  - *pre: at(M, from), level(M, Low)*
  - *eff: at(M, to),  $\neg$ at(M, from)*
- *Push(object, from, to)*
  - *pre: at(object, from), at(M, from), clear(object)*
  - *eff: at(M, to), at(object, to),  $\neg$ at(object, from),  $\neg$ at(M, from)*

# Operator example

---

- *Climb(object, p)*
  - *pre: at(M, p), at(object, p), level(M, Low), clear(object)*
  - *eff: level(M, High),  $\neg$ level(M, Low)*
- *ClimbDown()*
  - *pre: level(M, High)*
  - *eff:  $\neg$ level(M, High), level(M, Low)*



# Plan search

# Plan search

---

- *Given a planning problem (start state, operator descriptions, goal)*
- *Run standard search algorithms to find plan*
- *Decisions: search state representation, neighborhood, search algorithm*

# Linear planner

---

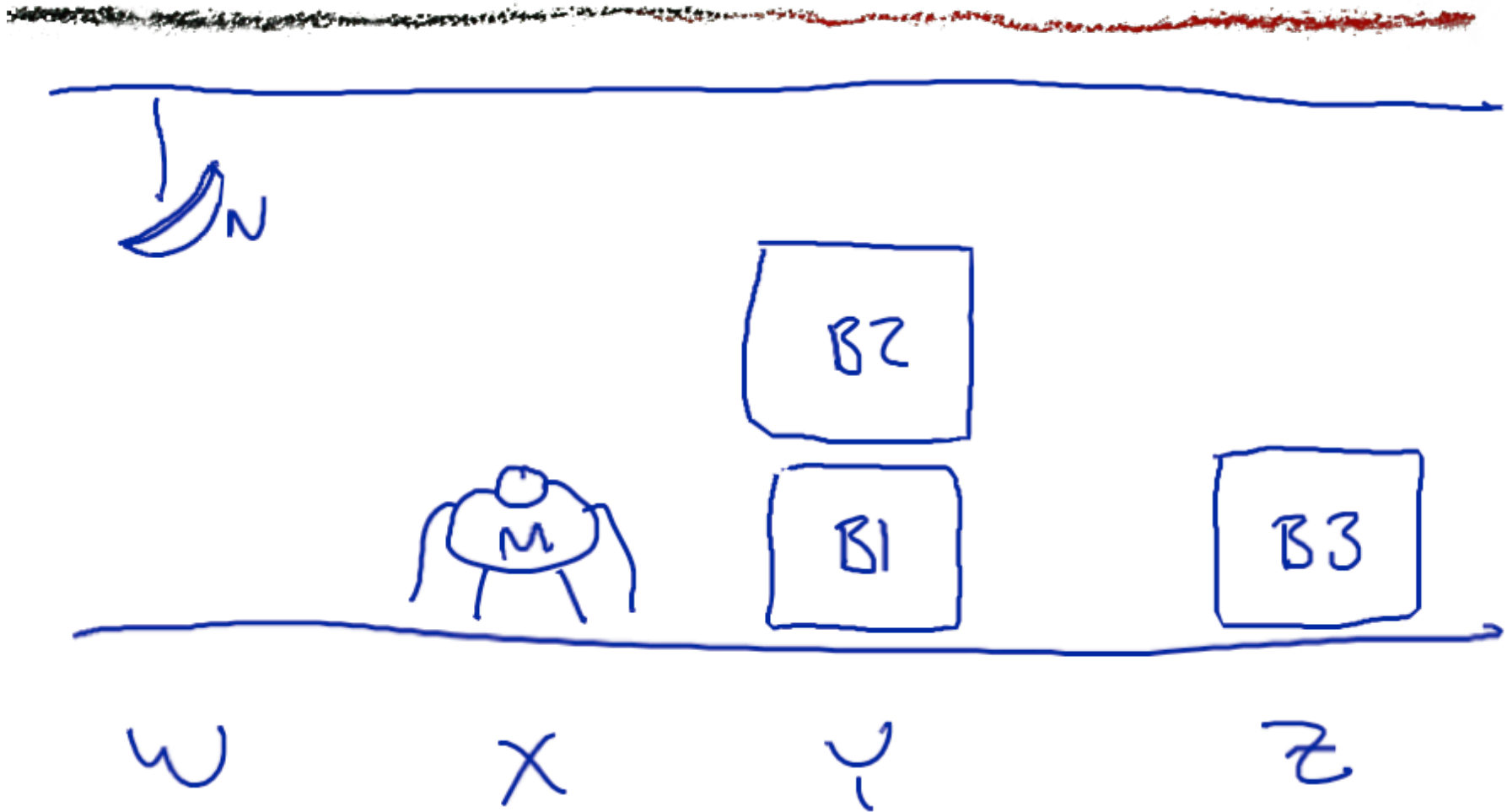
- *Simplest choice: linear planner*
- *Search state = sequence of operators*
- *Neighbor: add an operator to end of sequence*
- *Bind variables as necessary*
  - *both operator and binding are choice points*

# Linear planner



- *Can search forward from start or backward from goal*
- *Or mix the two*
- *Goal is often incompletely specified*
- *Example heuristic: number of open literals*

# Goal: full(M)



# STRIPS state example

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, X)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low)*
- *level(N, High)*

# Linear planner example

---

- *Start w/ empty plan [], initial world state*
- *Pick an operator, e.g.,*
  - *Move(from, to)*
    - *at(M, from), level(M, Low)*
    - *at(M, to),  $\neg$ at(M, from)*

# Linear planner example

---

- *Bind variables so that preconditions match world state*
  - *e.g., from: X, to: Y*
  - *pre: at(M, X), level(M, Low)*
  - *post: at(M, Y),  $\neg$ at(M, X)*

# Apply operator

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, X)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low)*
- *level(N, High)*

# Apply operator

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low)*
- *level(N, High)*

# Apply operator

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, Y)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low)*
- *level(N, High)*

# Repeat...

---

- *Plan is now [ move( $X$ ,  $Y$ ) ]*
- *World state is as in previous slide*
- *Pick another operator and binding*
  - *Climb(object,  $p$ ),  $p: Y$* 
    - *at( $M$ ,  $p$ ), at(object,  $p$ ), level( $M$ , Low), clear(object)*
    - *level( $M$ , High),  $\neg$ level( $M$ , Low)*

# Apply operator

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, Y)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, Low)*
- *level(N, High)*

# Apply operator

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, Y)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(N, High)*

# Apply operator

---

- *food(N)*
- *hungry(M)*
- *at(N, W)*
- *at(M, Y)*
- *at(B1, Y)*
- *at(B2, Y)*
- *at(B3, Z)*
- *on(B2, B1)*
- *clear(B2)*
- *clear(B3)*
- *level(M, High)*
- *level(N, High)*

# And so forth

---

- *Goal: full(M)*
- *A possible plan:*
  - *move(X, Y), move(Y, Z), push(B3, Z, Y),  
push(B3, Y, X), push(B3, X, W),  
climb(B3, W), eat(N, W, High)*
- *DFS will try moving XYX, climbing on  
boxes unnecessarily, etc.*

# Partial-order planner

---

- *Linear planner can be wasteful: backtrack undoes most recent action, rather than one that might have caused failure*
- *Partial order planner tries to fix this*
- *Avoids committing to details of plan until it has to (principle of least commitment)*

# Partial-order planner



- *Search state:*
  - *set of operators (partially bound)*
  - *ordering constraints*
  - *causal links (also called **guards**)*
  - *open preconditions*

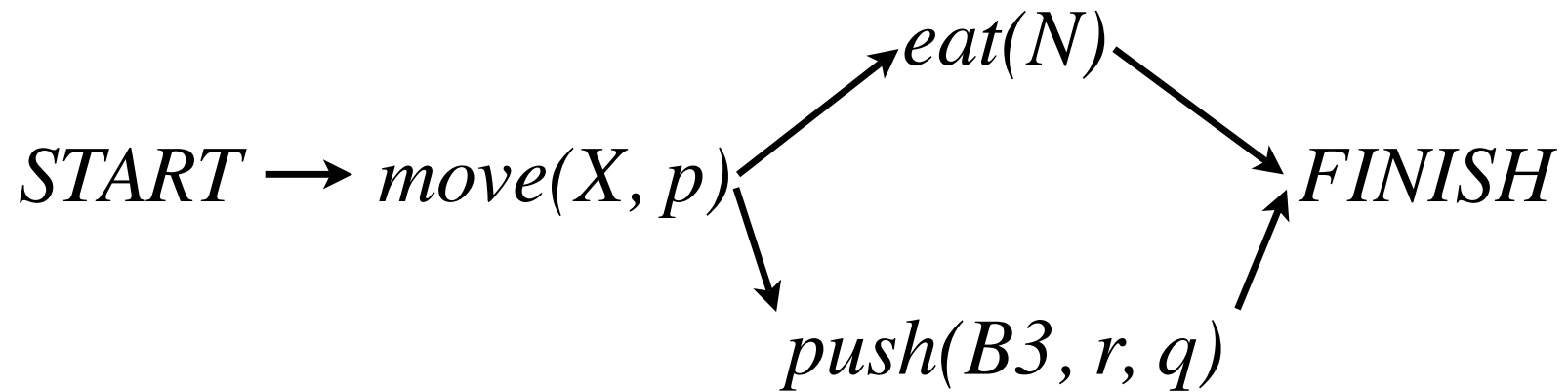
# Set of operators

---

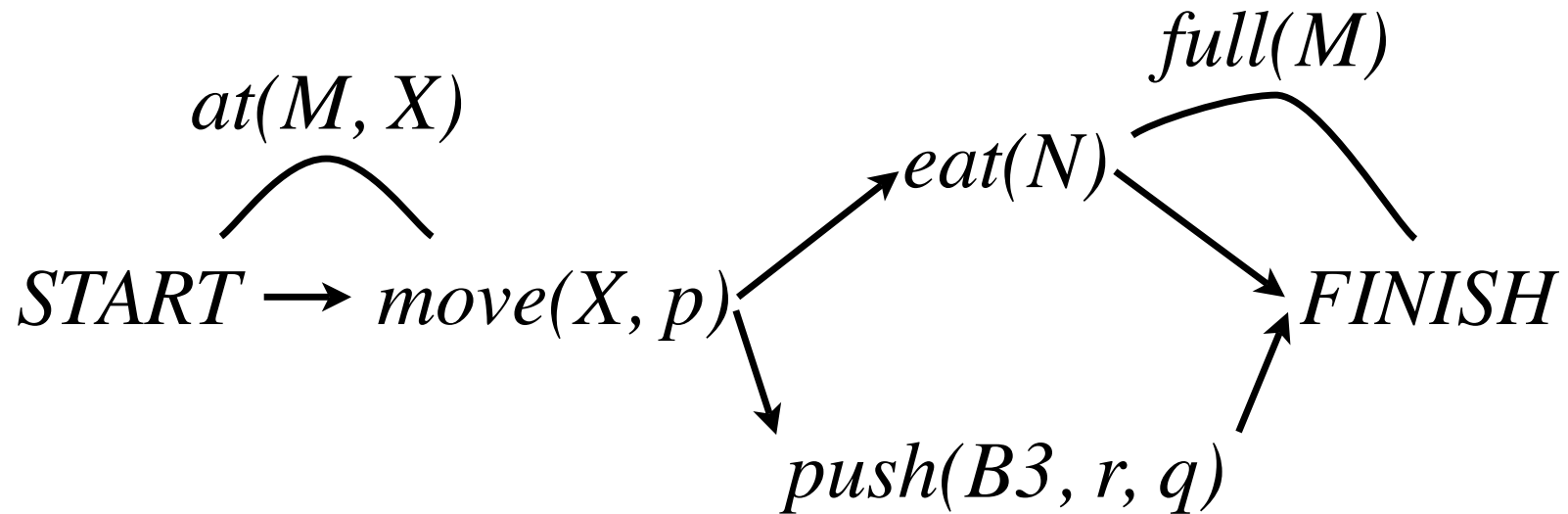
- *Might include  $move(X, p)$  “I will move somewhere from  $X$ ”,  $eat(target)$  “I will eat something”*
- *Also includes extra operators  $START$ ,  $FINISH$* 
  - *effects of  $START$  are initial state*
  - *preconditions of  $FINISH$  are goals*

# Partial ordering

---

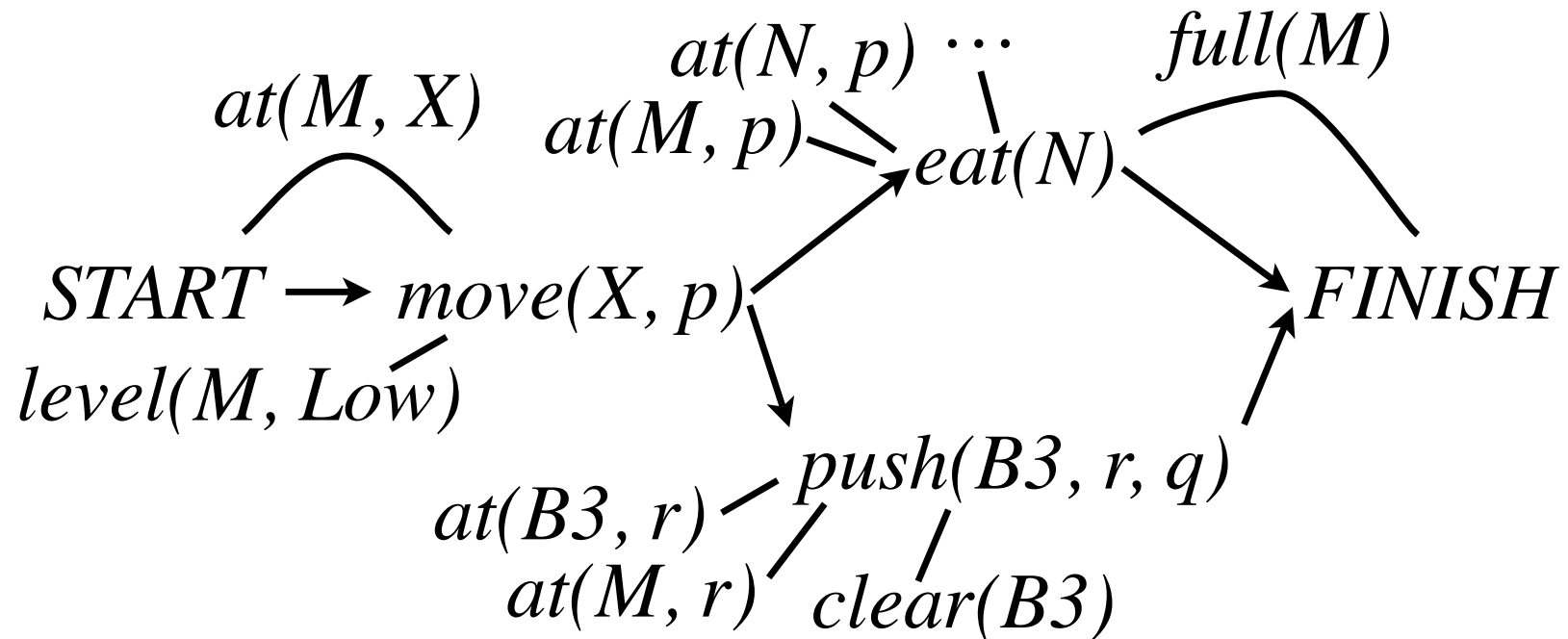


# Guards



- Describe where preconditions are satisfied

# Open preconditions



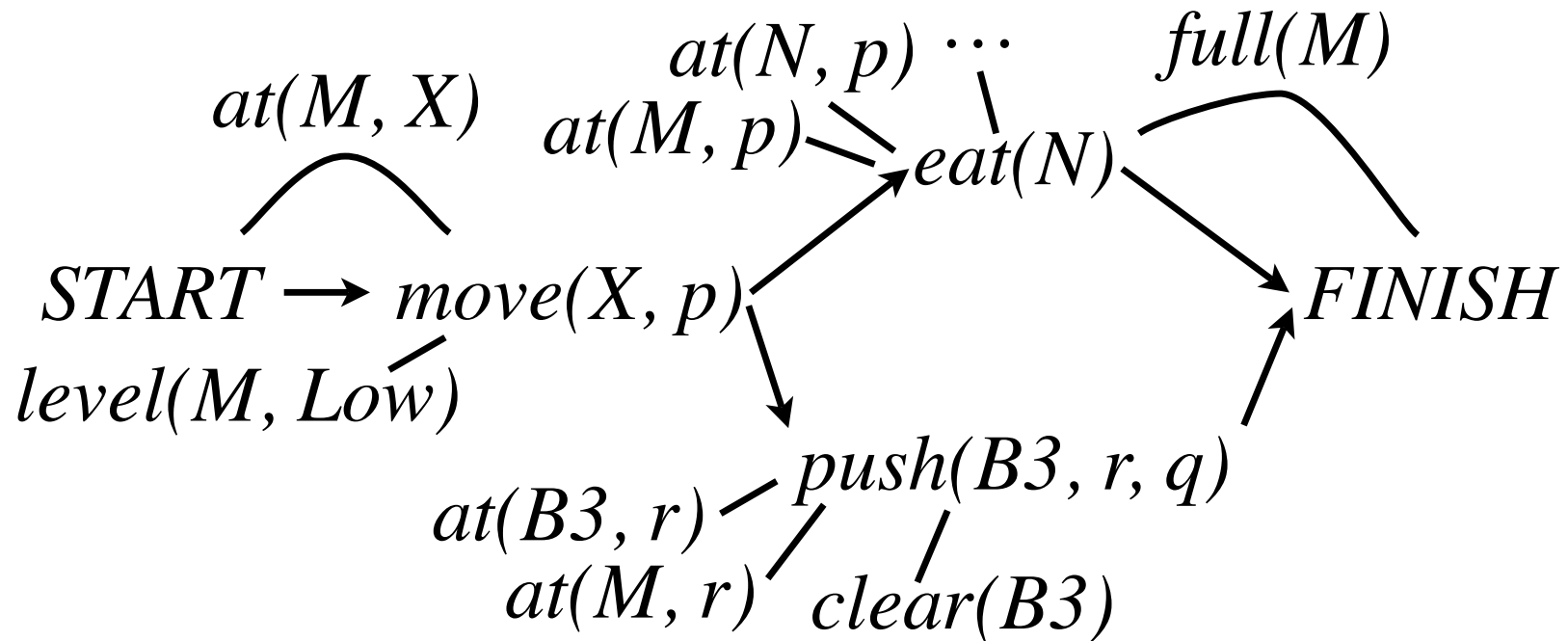
- *All unsatisfied preconditions of any action*
- *Unsatisfied = doesn't have a guard*

# Partial-order planner

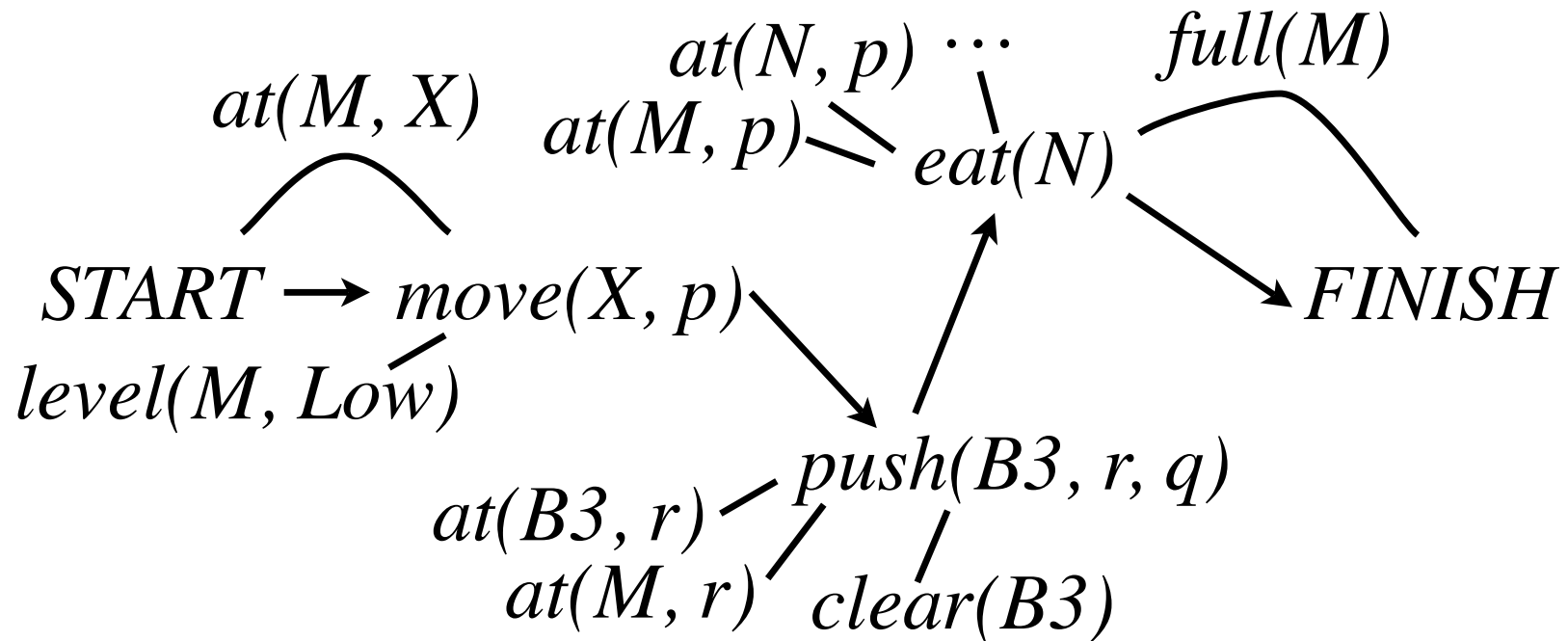


- *Neighborhood: plan refinement*
- *Add an operator, guard, or ordering constraint*

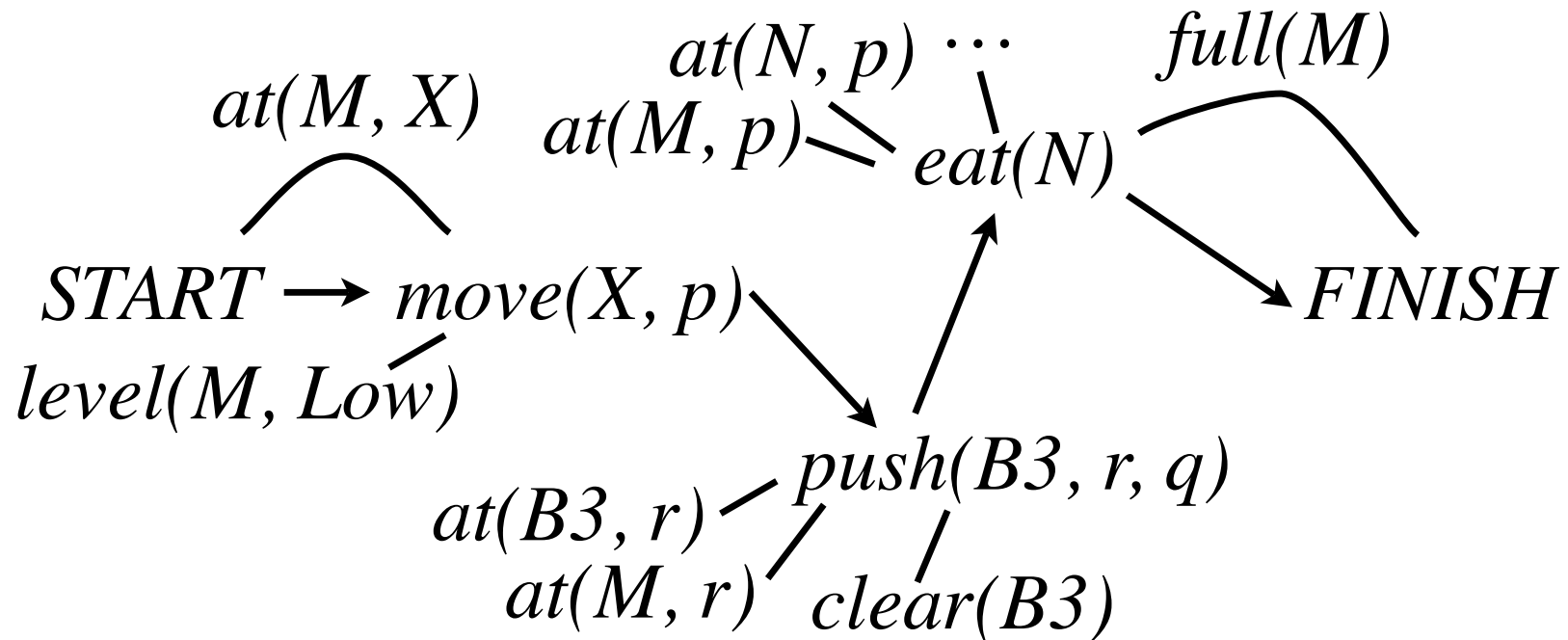
# Adding an ordering constraint



# Adding an ordering constraint

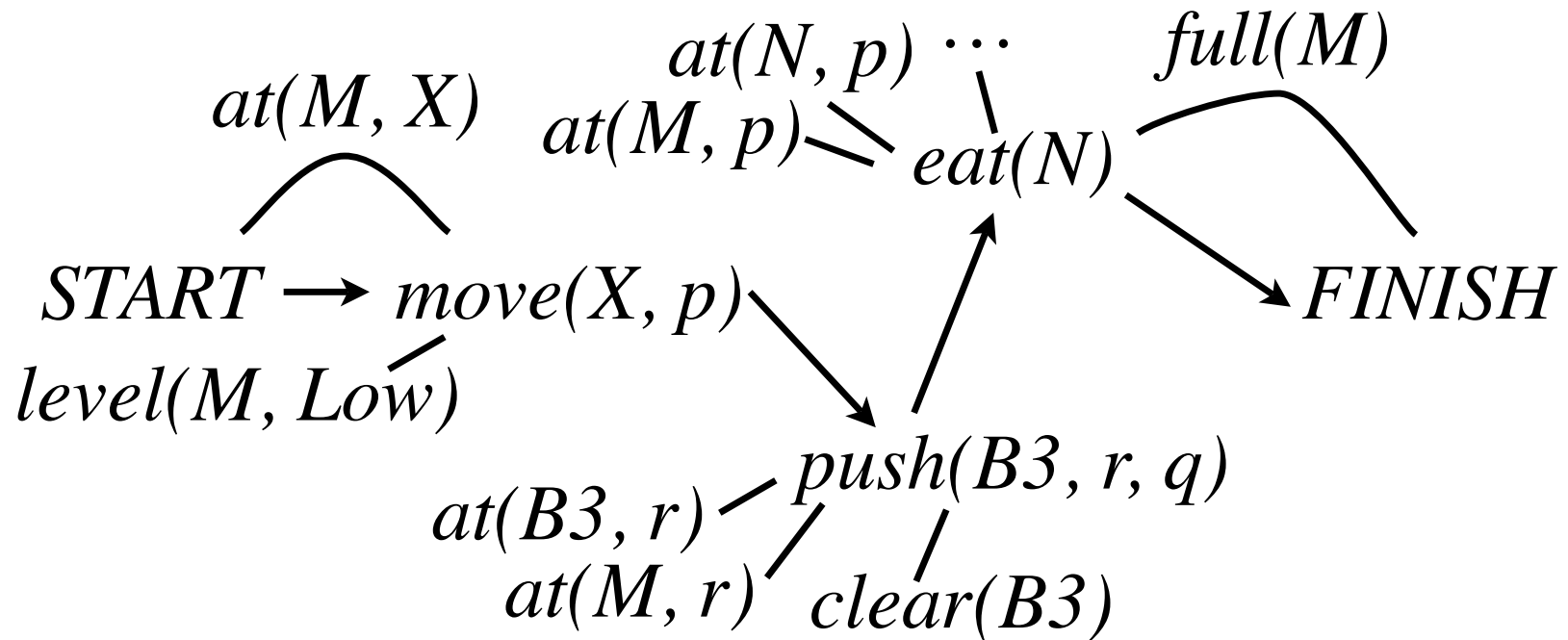


# Adding an ordering constraint

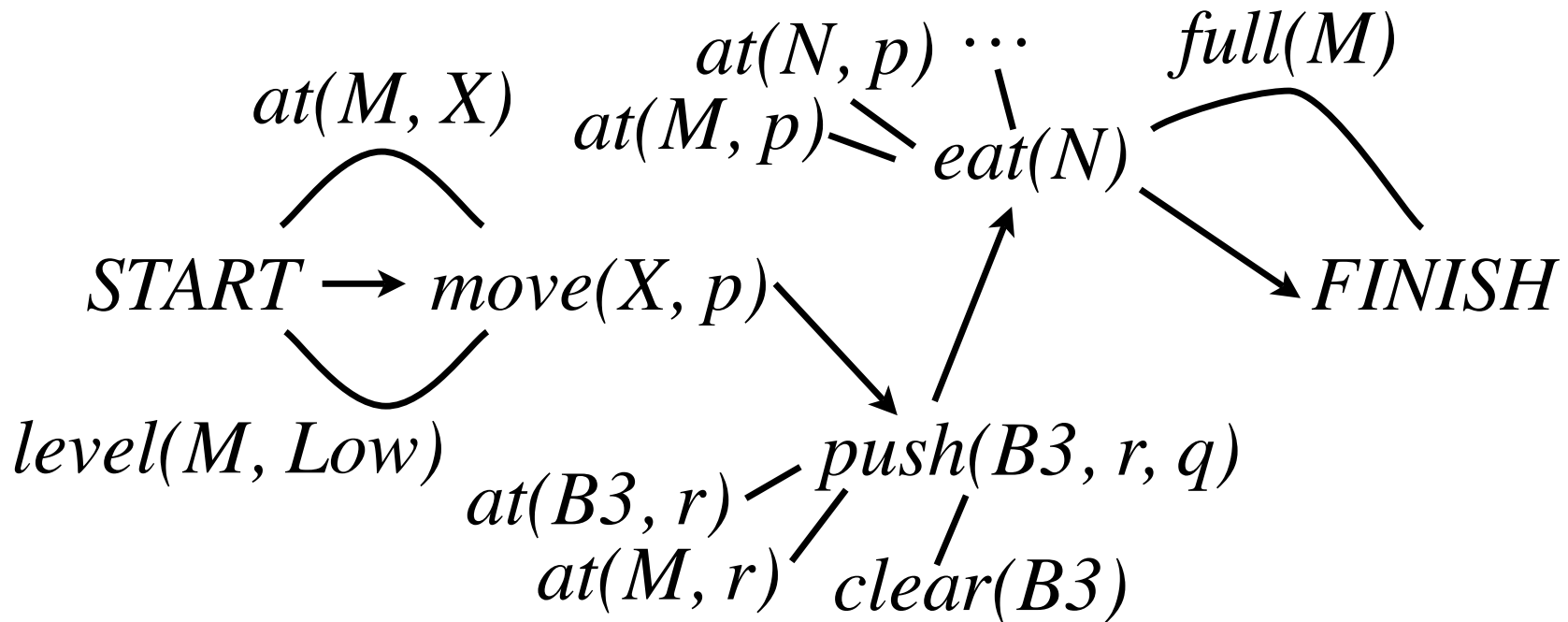


- *Wouldn't ever add ordering on its own—but may need to when adding operator or guard*

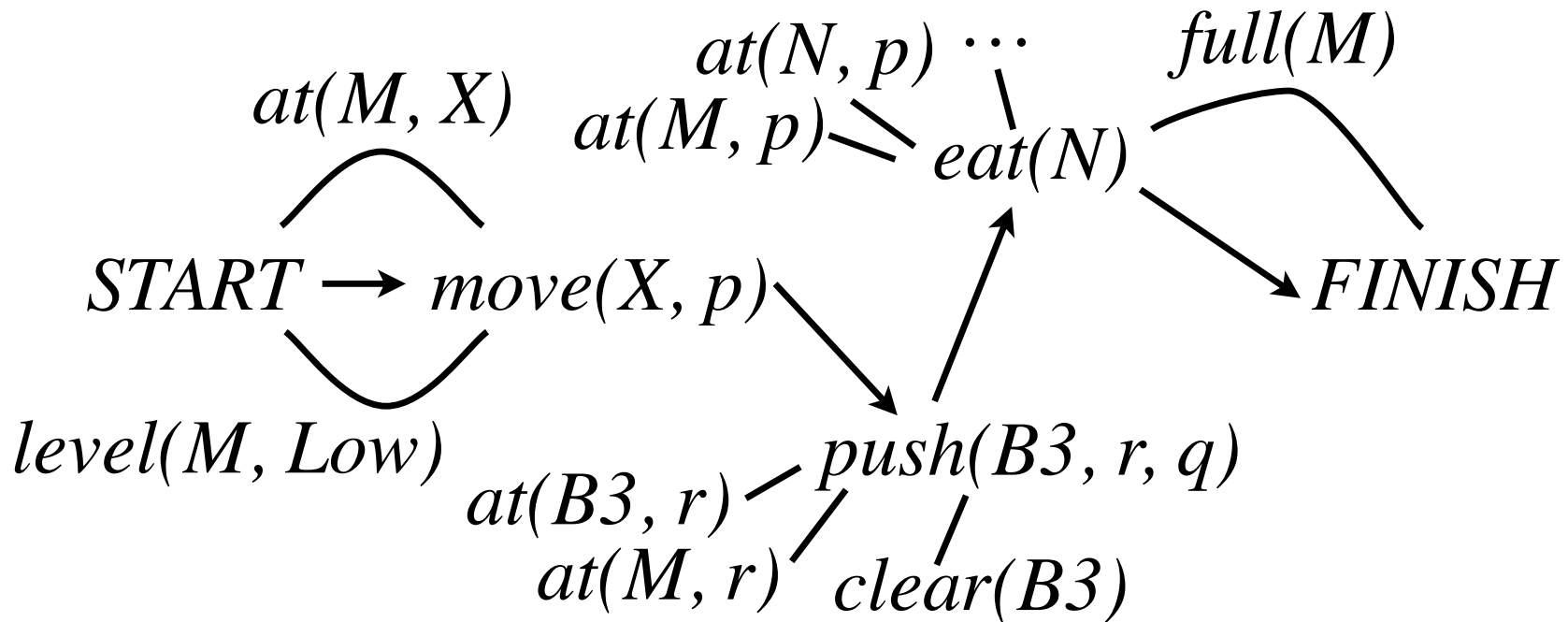
# Adding a guard



# Adding a guard

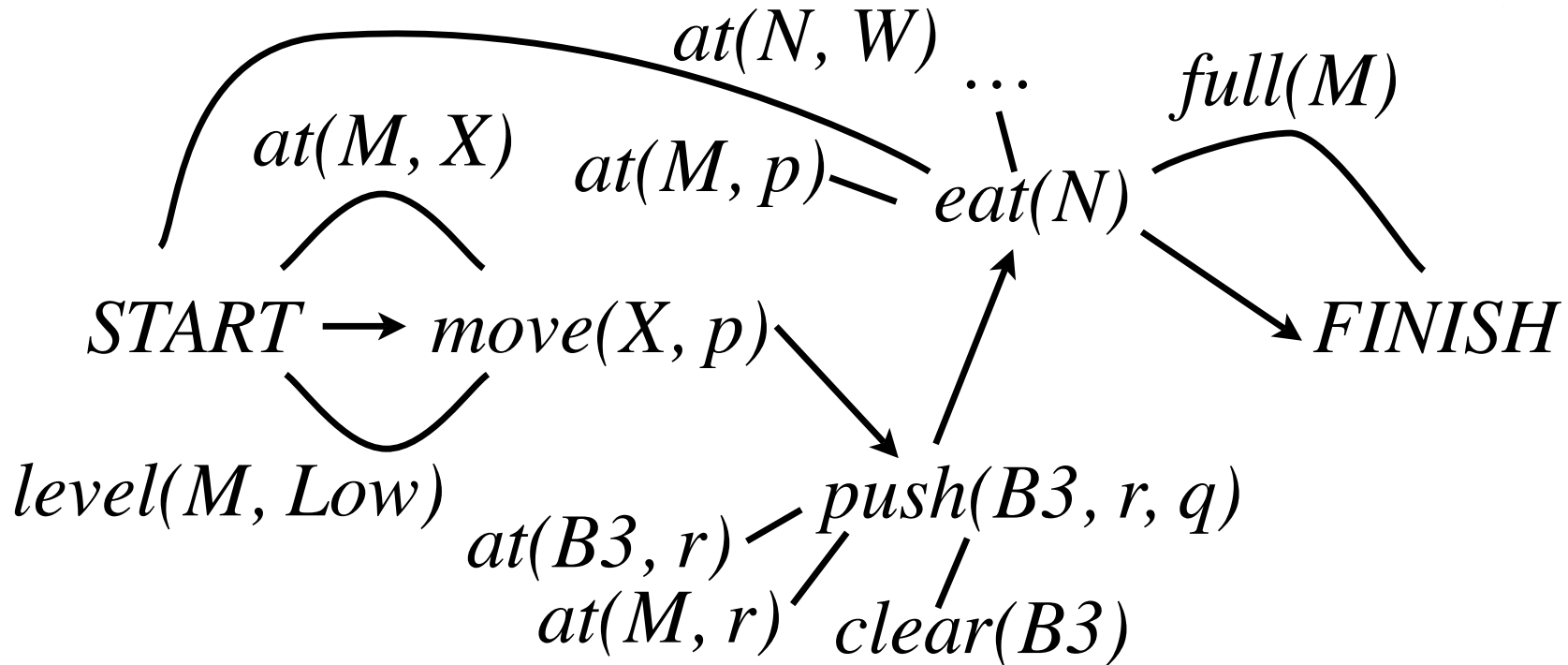


# Adding a guard



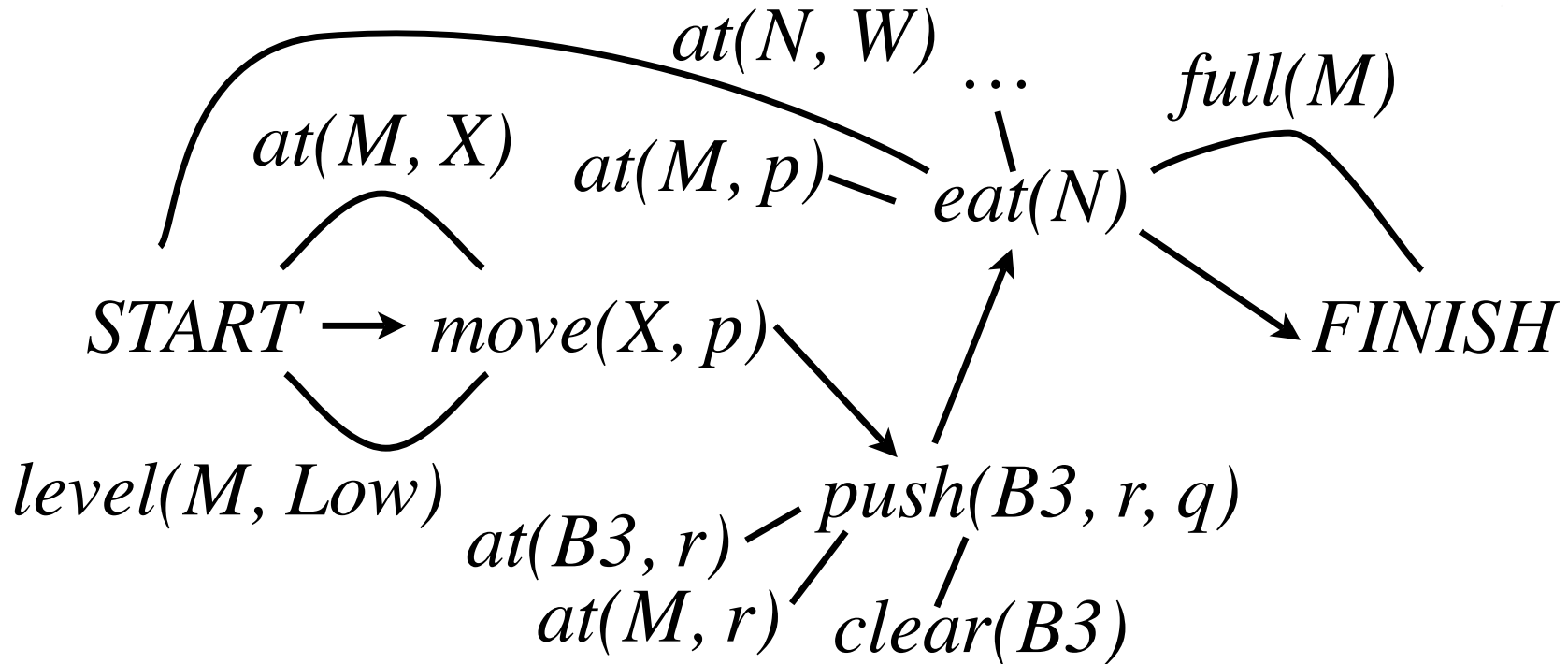
- *Must go forward (may need to add ordering)*
- *Can't cross operator that affects condition*

# Adding a guard

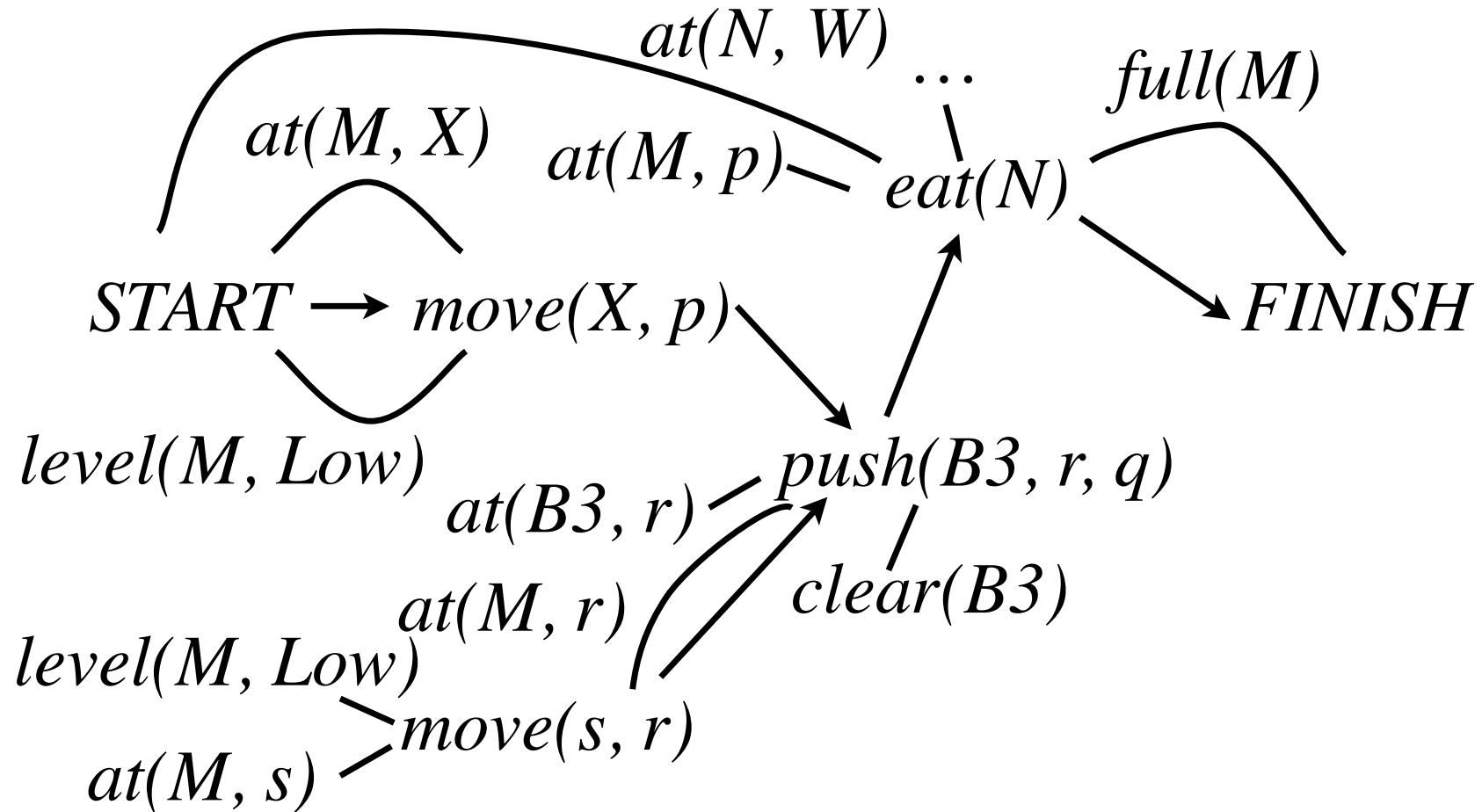


- *Might involve binding a variable (may be more than one way to do so)*

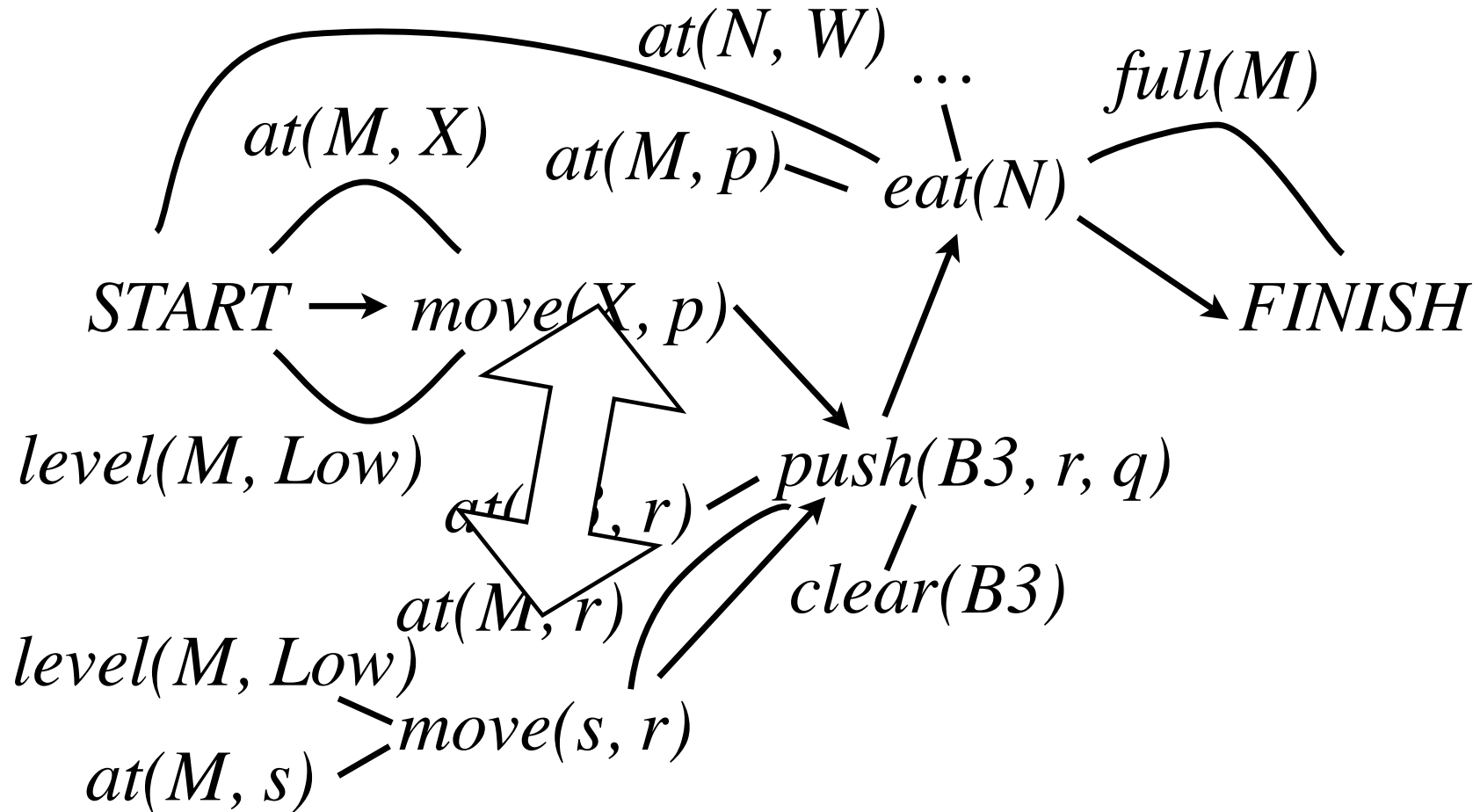
# Adding an operator



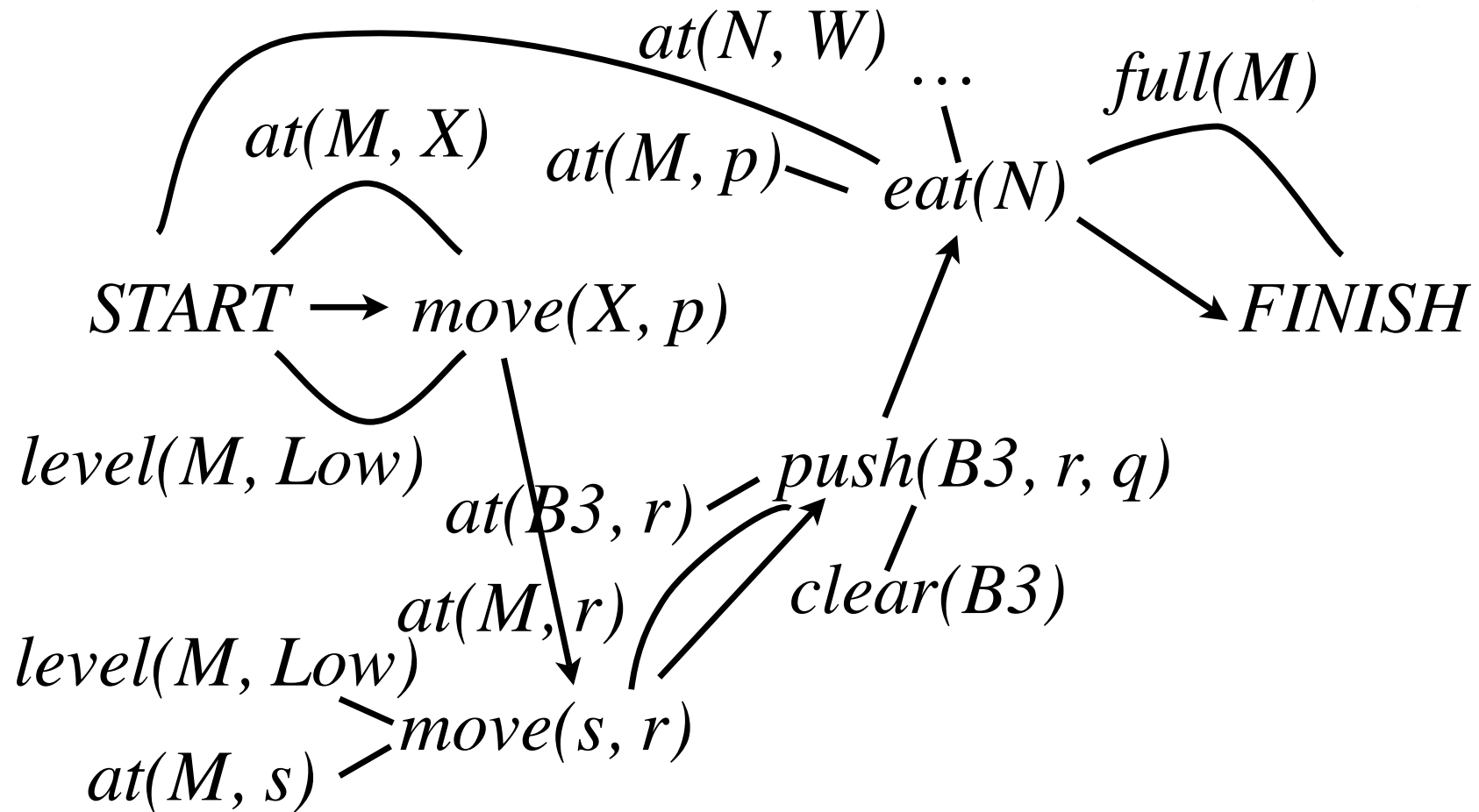
# Adding an operator



# Adding an operator



# Resolving conflict



# Recap of neighborhood

---

- *Pick an open precondition*
- *Pick an operator and binding that can satisfy it*
  - *may need to add a new op*
  - *or can use existing op*
- *Add an ordering constraint and guard*
- *Resolve conflicts by adding more ordering constraints or bindings*

# Consistency & completeness

---

- *Consistency: no cycles in ordering, preconditions guaranteed true throughout guard intervals*
- *Completeness: no open preconditions*
- *Search maintains consistency, terminates when complete*

# Execution



- *A consistent, complete plan can be executed by **linearizing** it*
- *Execute actions in any order that matches the ordering constraints*
- *Fill in unbound variables in any consistent way*



# Plan Graphs

# Planning & model search

---

- *For a long time, it was thought that SAT-style model search was a non-starter as a planning algorithm*
- *More recently, people have written fast planners that*
  - *propositionalize the domain*
  - *turn it into a CSP or SAT problem*
  - *search for a model*

# Plan graph

---

- *Tool for making good CSPs: plan graph*
- *Encodes a subset of the constraints that plans must satisfy*
- *Remaining constraints are handled during search (by rejecting solutions that violate them)*

# Example

---

- *Start state: have(Cake)*
- *Goal: have(Cake)  $\wedge$  eaten(Cake)*
- *Operators: bake, eat*

# Operators



- *Bake*
  - *pre:  $\neg$ have(Cake)*
  - *post: have(Cake)*
- *Eat*
  - *pre: have(Cake)*
  - *post:  $\neg$ have(Cake), eaten(Cake)*

# Propositionalizing



- *Note: this domain is fully propositional*
- *If we had a general STRIPS domain, would have to pick a universe and propositionalize*
- *E.g.,  $eat(x)$  would become  $eat(Banana)$ ,  $eat(Cake)$ ,  $eat(Fred)$ , ...*

# Plan graph



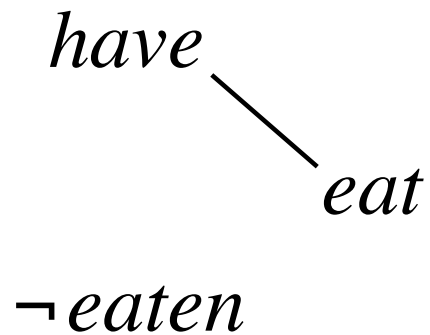
*have*

$\neg$  *eaten*

- *Alternating levels: states and actions*
- *First level: initial state*

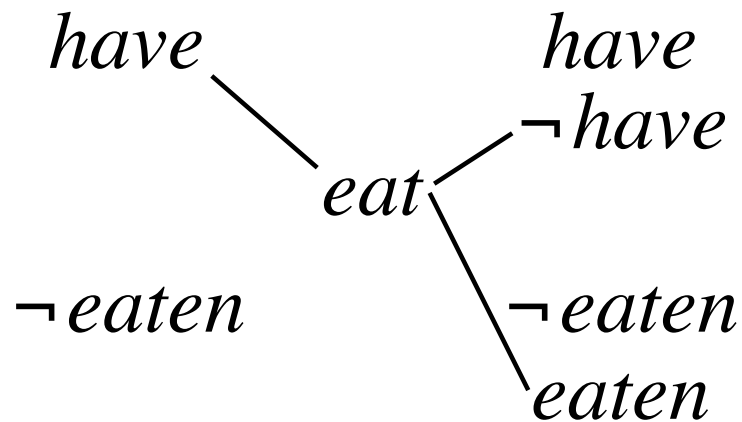
# Plan graph

---



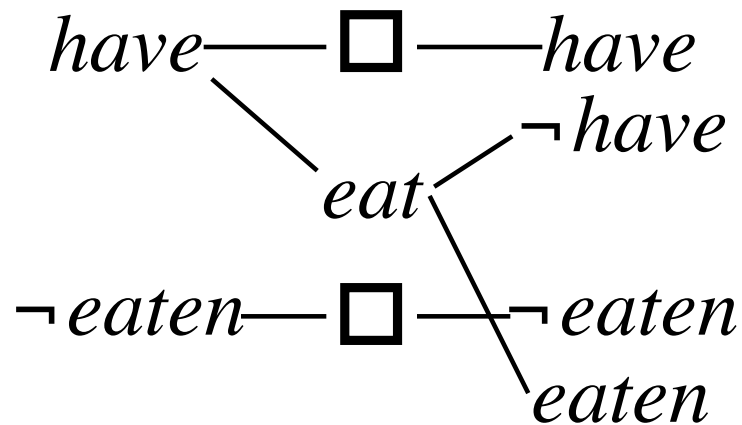
- *First action level: all applicable actions*
- *Linked to their preconditions*

# Plan graph



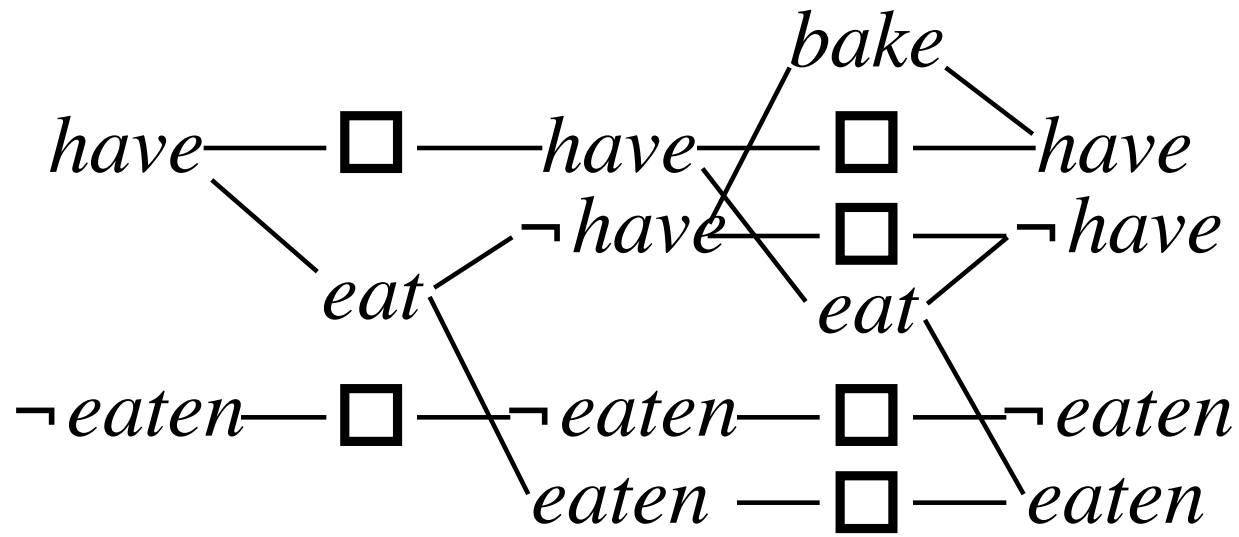
- *Second state level: add effects of actions to get literals that could hold at step 2*

# Plan graph



- *Also add maintenance actions to represent effect of doing nothing*

# Plan graph



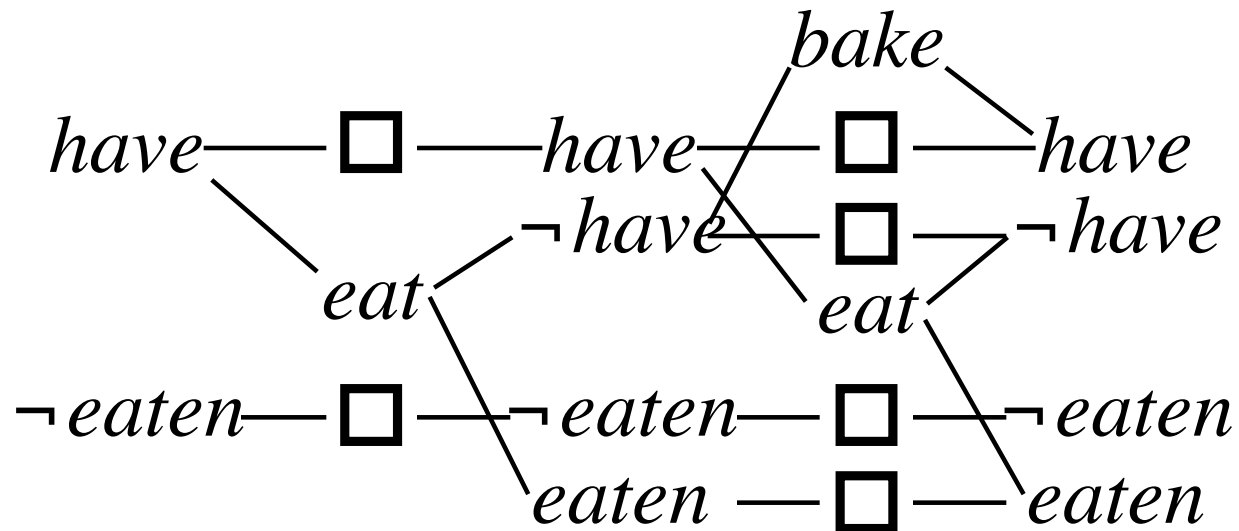
- *Extend another pair of levels: now bake is a possible action*

# Plan graph



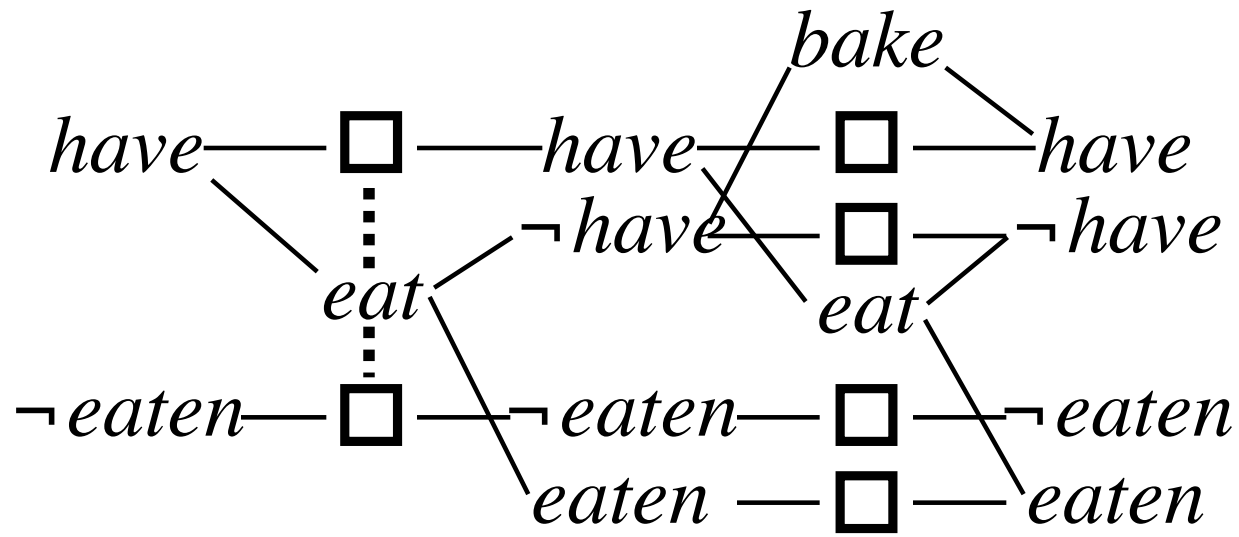
- *Can extend as far right as we want*
- *Plan = subset of the actions at each action level*
- *Ordering unspecified within a level*

# Plan graph



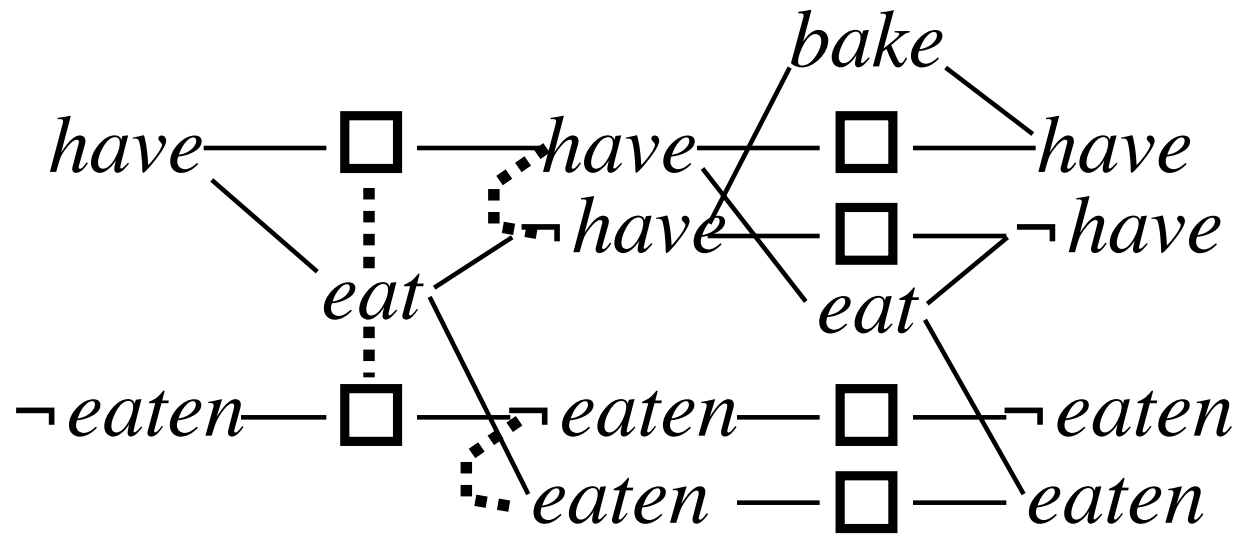
- *In addition to the above links, add **mutex** links to indicate mutually exclusive actions or literals*

# Plan graph



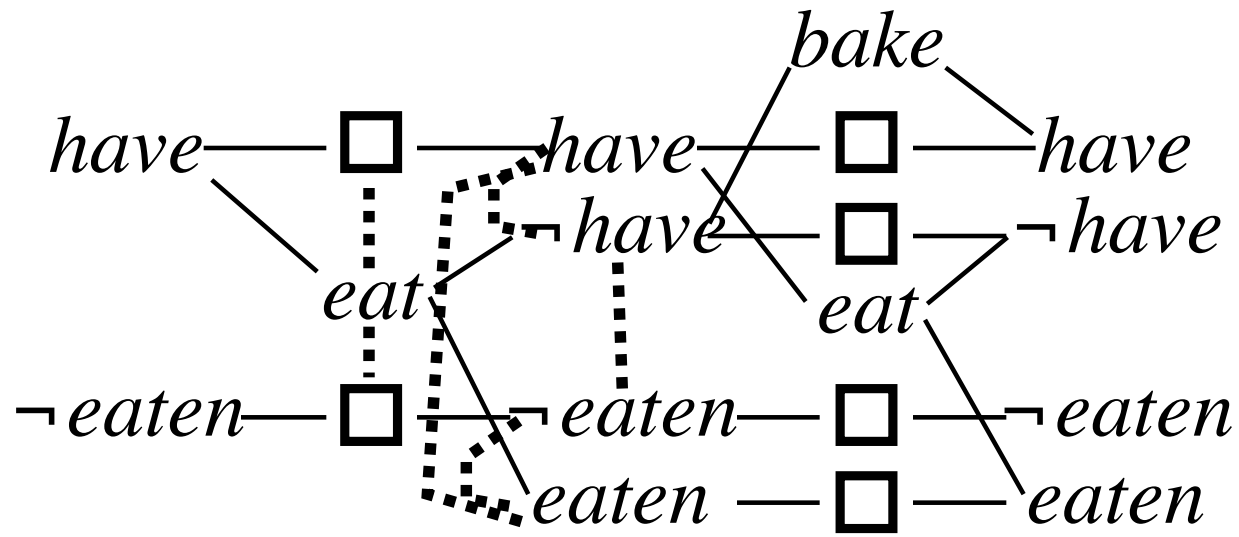
- *Actions which assert contradictory literals are mutex*

# Plan graph



- *Literals are mutex if they are contradictory*

# Plan graph



- *Or if there is no non-mutex set of actions that could achieve both*



# Getting a plan

---

- *Build the plan graph out to some length  $k$*
- *Translate to a SAT formula or CSP*
- *Search for a satisfying assignment*
- *If found, read off the plan*
- *If not, increment  $k$  and try again*
- *There is a test to see if  $k$  is big enough*

# Translation to SAT

---

- *One variable for each pair of literals in state levels*
- *One variable per action in action levels*
- *Constraints implement STRIPS semantics*
- *Solution tells us which actions are performed at each action level, which literals are true at each state level*

# Action constraints

---

- *Each action can only be executed if all of its preconditions are present:*

$$act_{t+1} \Rightarrow pre1_t \wedge pre2_t \wedge \dots$$

- *If executed, action asserts its postconditions:*

$$act_{t+1} \Rightarrow post1_{t+2} \wedge post2_{t+2} \wedge \dots$$

# Literal constraints



- *In order to achieve a literal, we must execute an action that achieves it*
  - $post_{t+2} \Rightarrow act1_{t+1} \vee act2_{t+1} \vee \dots$
- *Might be a maintenance action*

# Initial & goal constraints

---

- *Goals must be satisfied at end:*

$$goal1_T \wedge goal2_T \wedge \dots$$

- *And initial state holds at beginning:*

$$init1_1 \wedge init2_1 \wedge \dots$$

# Mutex constraints

---

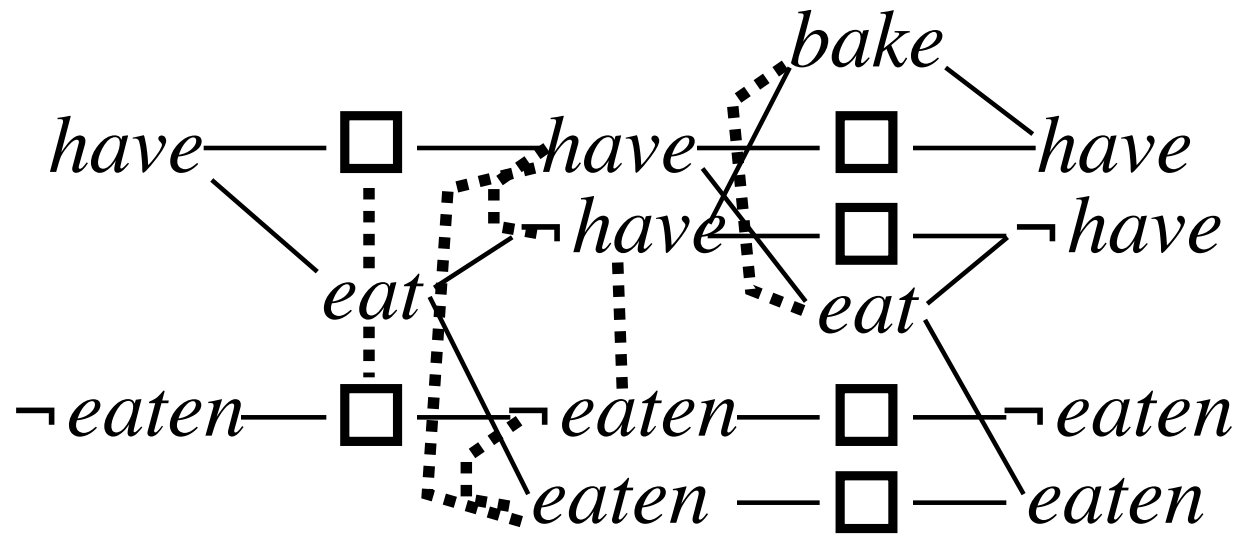
- *Mutex constraints between actions or literals: add clause  $(x \oplus y)$*
- *Note: some mutexes are redundant, but help anyway*

# Plan search

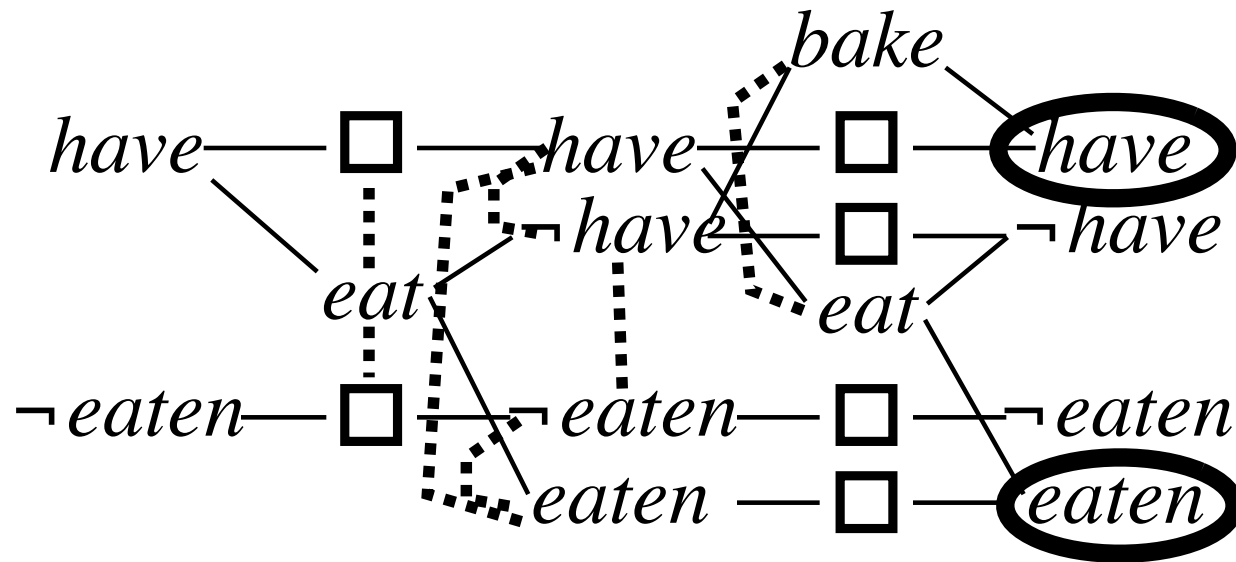
---

- *Hand problem to SAT solver*
- *Or, simple DFS: start from last level, fill in last action set, compute necessary preconditions, fill in 2nd-to-last action set, etc.*
- *If at some level there is no way to do any actions, or no way to fill in consistent preconditions, backtrack*

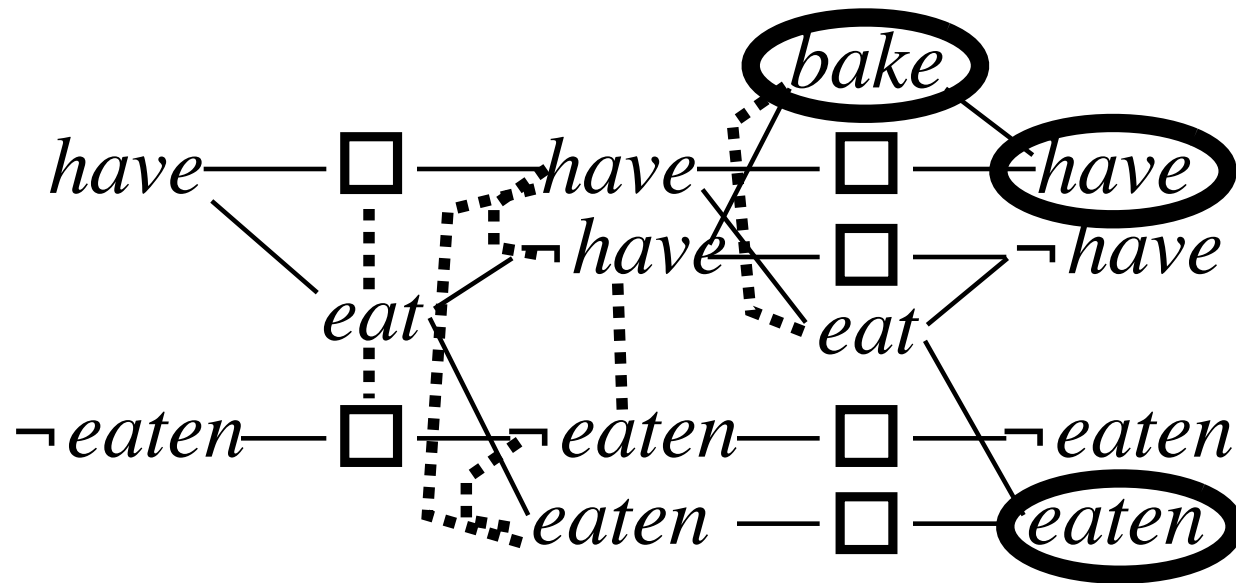
# Plan search



# Plan search

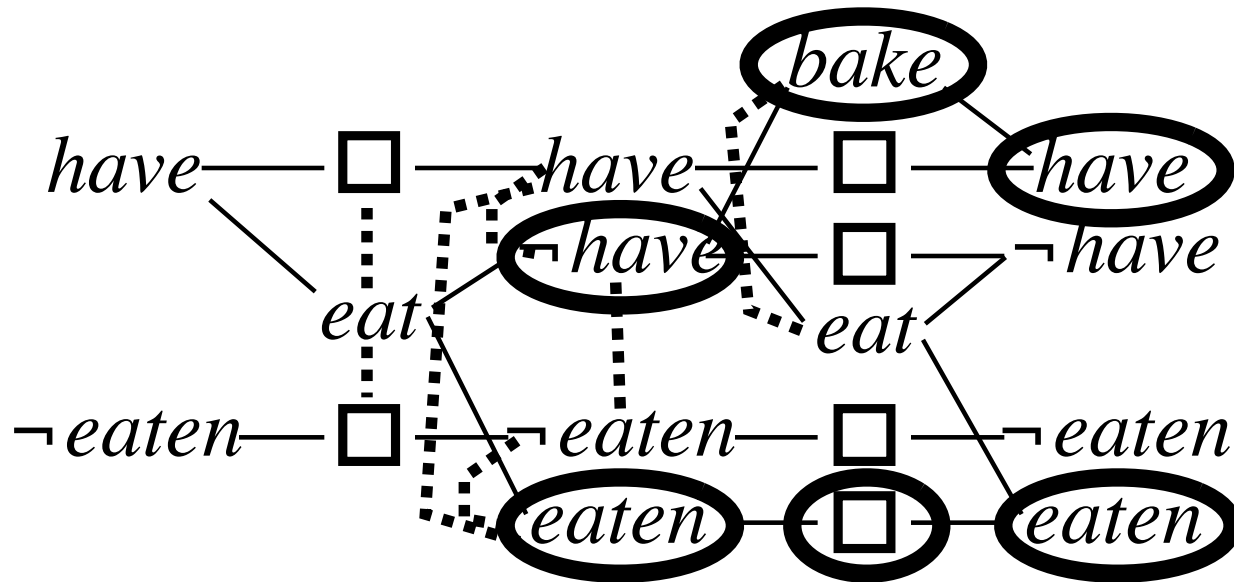


# Plan search





# Plan search





# Plan search

