Planning-Graph and Satisfiability Techniques

Chapter 10

Outline

♢ Planning graph techniques: ← subtle

- Motivation
- Planning graph
- Mutual exclusion
- Plan extraction
- Example

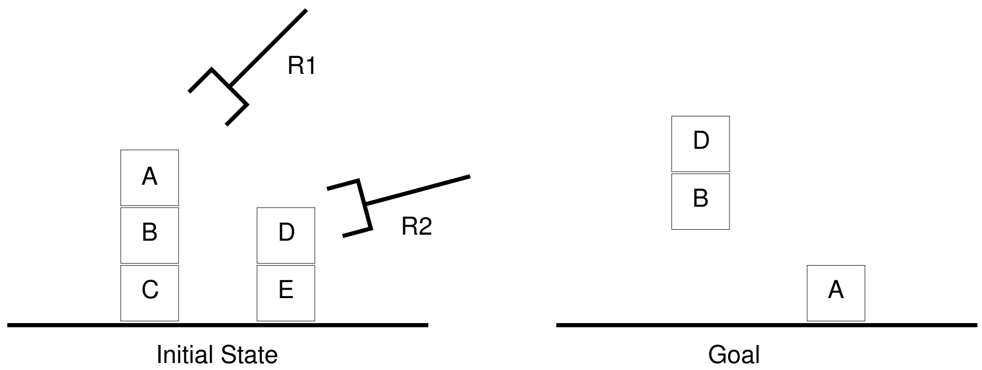
♢ SAT planning techniques: ← much easier

- Motivation
- Variables and clauses
- Example
- Encoding improvements
- Evaluation strategies

Motivation

State-space search produces inflexible plans.

Part of the ordering in an action sequence is irrelevant. We only want to order actions to reflect positive or negative interactions between actions.

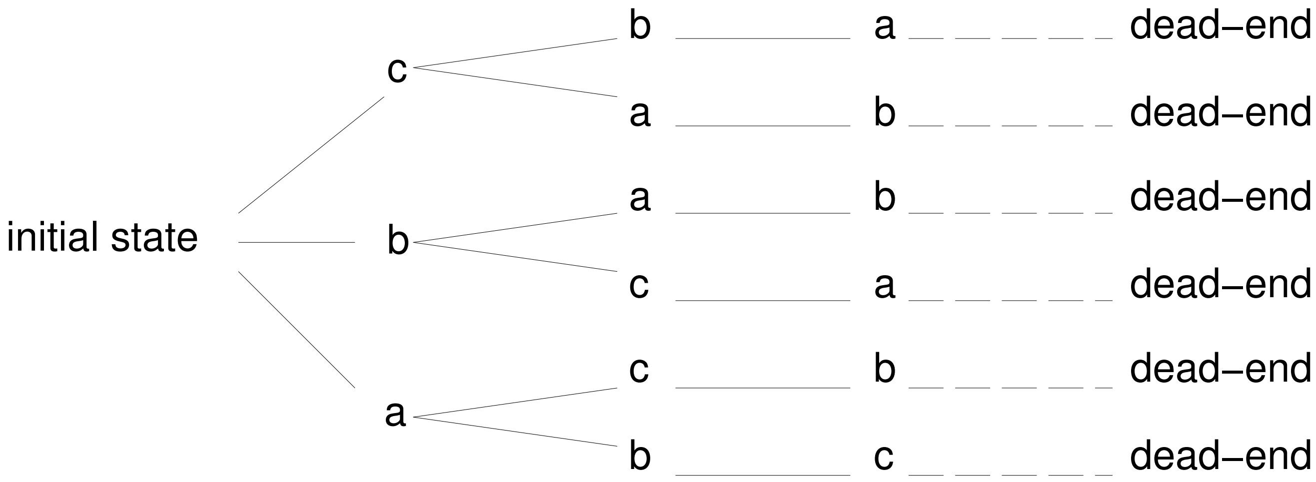


sequence:
$\langle \text{unstack}(R1, A, B), \text{unstack}(R2, D, E), \text{putdown}(R1, A), \text{stack}(R2, D, B) \rangle$

parallel plan:
$\langle \{\text{unstack}(R1, A, B), \text{unstack}(R2, D, E)\}, \{\text{putdown}(R1, A), \text{stack}(R2, D, B)\} \rangle$

Motivation

State-space search wastes time examining many different orderings of the same set of actions:



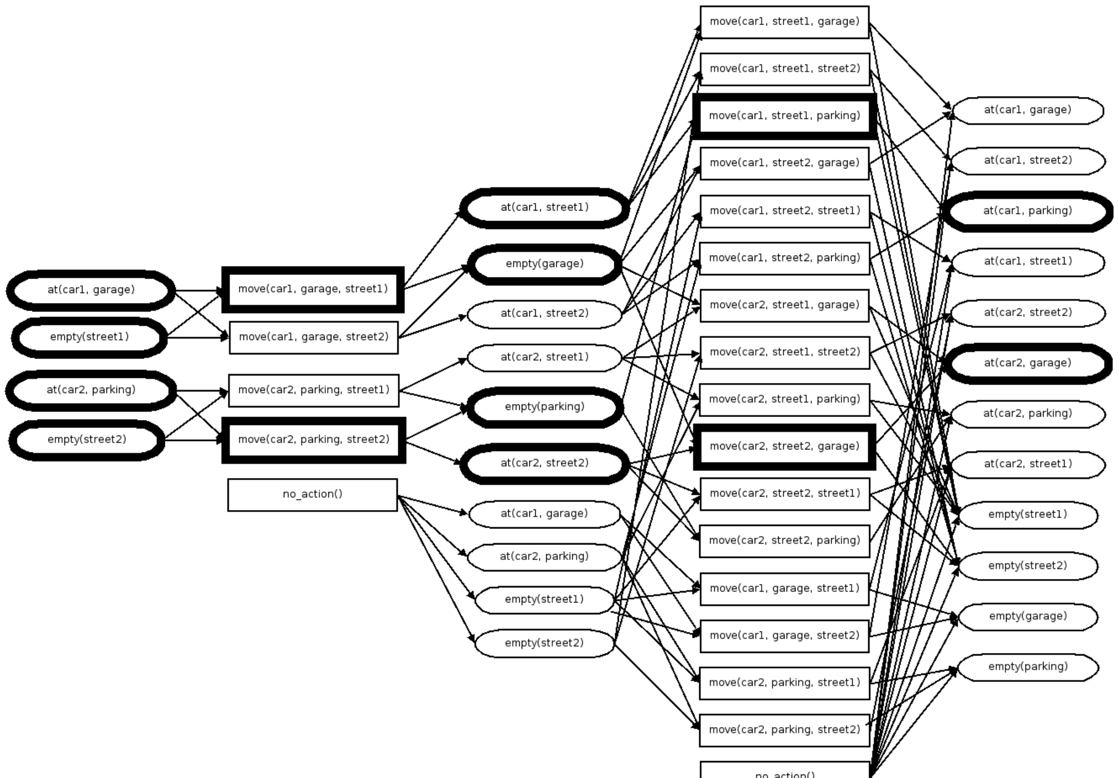
Not ordering actions that can take place in parallel can speed up planning

Motivation

State space search does not exploit the structure of states (except for heuristics)

An alternative would be to reason about the propositions that can be true and about the actions that can be applicable after 1, 2, . . . , k plan steps.

Amounts to relaxing the state space by "unioning' some of the states and propagating positive and negative interaction constraints.



Graphplan

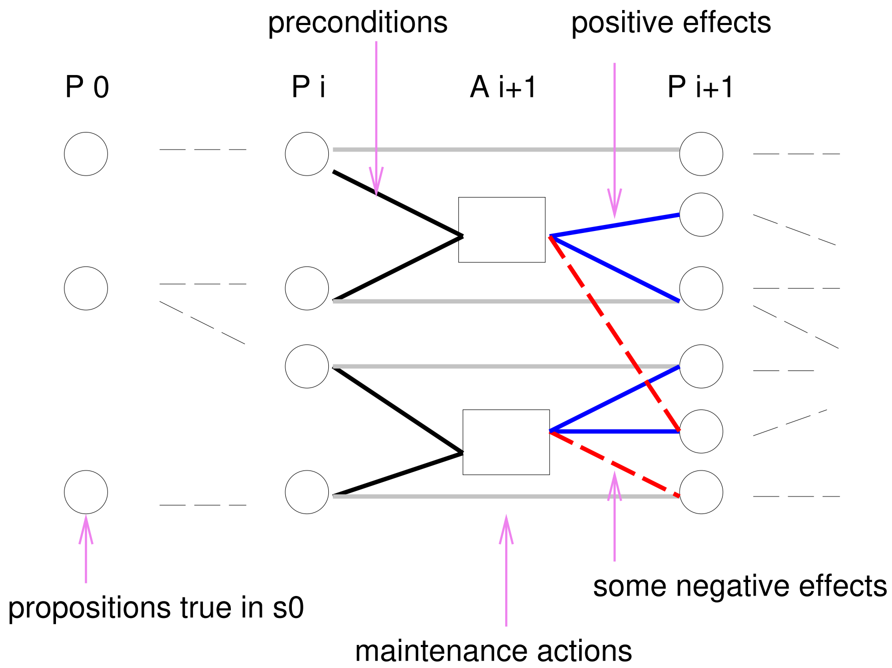
This is how Graphplan [Blum & First, IJCAI 1995] implements this idea:

1. Build a planning graph which can be viewed as a relaxation of the state space over k steps (note: a step includes several parallel actions). This can be done in polynomial time.
2. The planning graph captures information about **pairs** of mutually exclusive actions and propositions. It gives us a **necessary** but insufficient condition for when the goal is reachable in k steps.
3. Attempt to extract a parallel plan from the graph using a form of backward search through the graph.
4. If the extraction is unsuccessful, k is incremented, the graph extended, and a new extraction performed, and so on, until a plan is found or we determine that the problem is unsolvable.

The planning graph

Alternating layers of propositions and actions, $P_0, A_1, P_1, \ldots, A_i, P_i, \ldots A_k, P_k$:

- $P_0 = s_0$
- A_{i+1} contains the actions that might be able to occur at time step $i+1$. Their preconditions must belong to P_i. We include maintenance actions (prec p, eff p) for each proposition $p \in P_i$ to represent what happens if no action at this layer in the final plan affects p.
- P_{i+1} contains the propositions that are **positive** effects of actions in A_{i+1}.

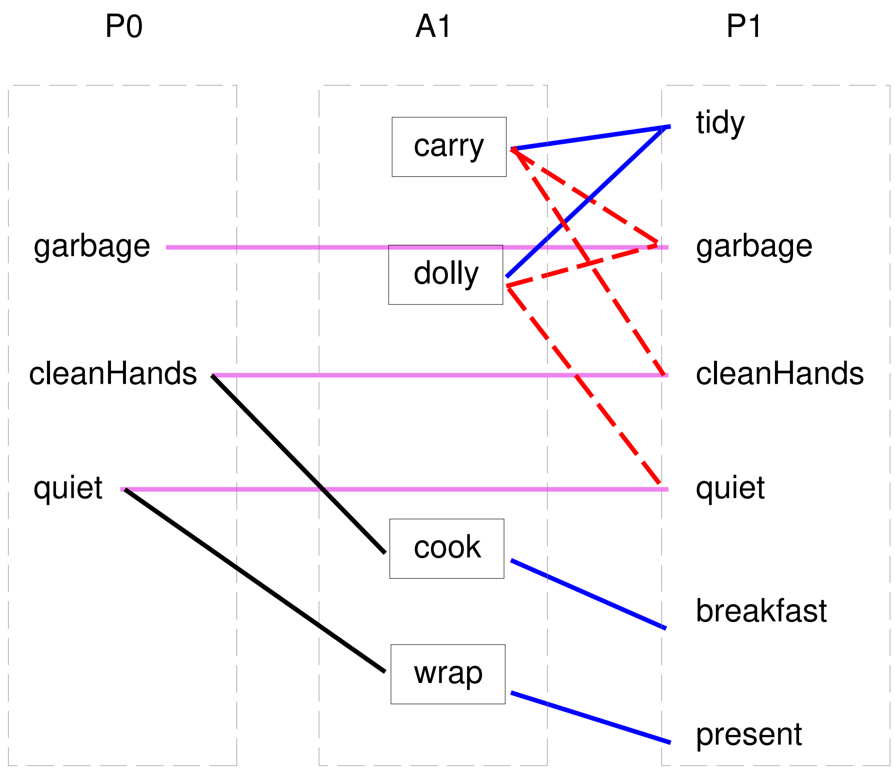


The planning graph: example

Suppose you want to make surprise for your mother who is asleep

Operator	Precondition	Effect
cook()	{cleanHands}	{breakfast}
wrap()	{quiet}	{present}
carry()	{}	{tidy, ¬garbage, ¬cleanHands}
dolly()	{}	{tidy, ¬garbage, ¬quiet}

$s_0 = \{\text{garbage, cleanHands, quiet}\}$
$g = \{\text{breakfast, present, tidy}\}$



Mutual exclusion

The plan graph records **limited** information about negative interactions

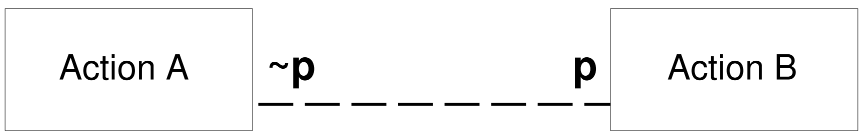
It records **pairs** of actions which cannot happen in parallel and pairs of propositions which cannot be simultaneously true. These are called mutex

The action parallelism notion underlying the mutex relation is independence:

> two actions are independent when executing them in any order
> is possible and yields the same result

For independence, we must avoid:

- interference: one action deletes a precondition of the other (one of the two orderings is not possible)
- inconsistence: one action deletes a positive effect of the other (the two orderings yield different results)



Action B not applicable

Mutual exclusion

The plan graph records **limited** information about negative interactions

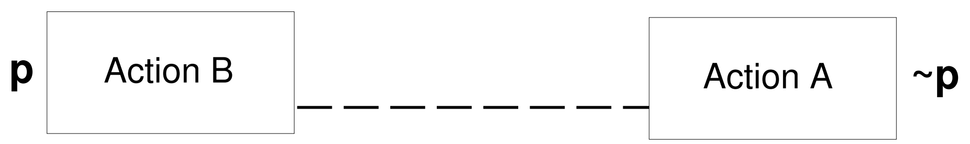
It records **pairs** of actions which cannot happen in parallel and pairs of propositions which cannot be simultaneously true. These are called mutex

The action parallelism notion underlying the mutex relation is independence:

> two actions are independent when executing them in any order
> is possible and yields the same result

For independence, we must avoid:

- interference: one action deletes a precondition of the other (one of the two orderings is not possible)
- inconsistence: one action deletes a positive effect of the other (the two orderings yield different results)



Action B applicable

Mutual exclusion

The plan graph records **limited** information about negative interactions

It records **pairs** of actions which cannot happen in parallel and pairs of propositions which cannot be simultaneously true. These are called mutex

The action parallelism notion underlying the mutex relation is independence:

> two actions are independent when executing them in any order is possible and yields the same result

For independence, we must avoid:

- interference: one action deletes a precondition of the other (one of the two orderings is not possible)
- inconsistence: one action deletes a positive effect of the other (the two orderings yield different results)

Mutual exclusion

The plan graph records **limited** information about negative interactions

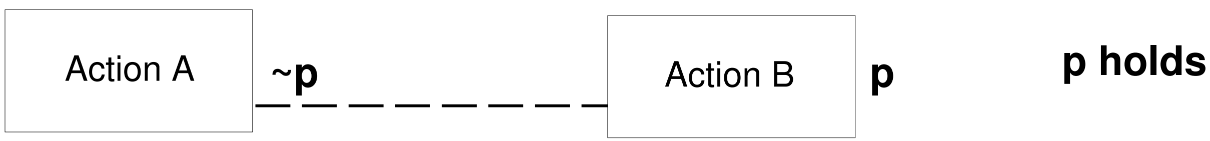
It records **pairs** of actions which cannot happen in parallel and pairs of propositions which cannot be simultaneously true. These are called mutex

The action parallelism notion underlying the mutex relation is independence:

> two actions are independent when executing them in any order
> is possible and yields the same result

For independence, we must avoid:

- interference: one action deletes a precondition of the other (one of the two orderings is not possible)
- inconsistence: one action deletes a positive effect of the other (the two orderings yield different results)



Mutual exclusion

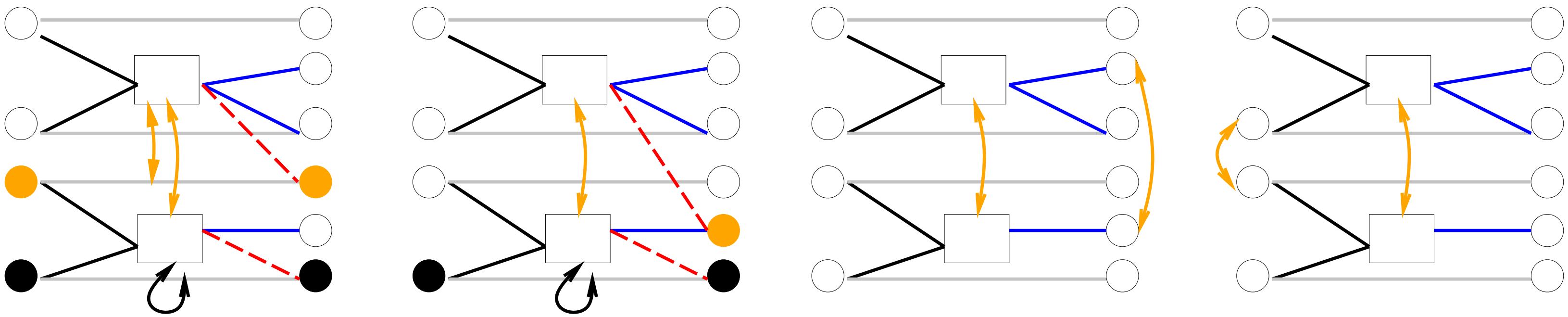
Two actions at the same level of the graph are mutex if they:

- interfere: one deletes a precondition of the other
- are inconsistent: one deletes a positive effect of the other
- have competing needs: they have mutually exclusive preconditions

Two propositions at the same level are mutex if they:

- have inconsistent support: all ways of achieving both are pairwise mutex

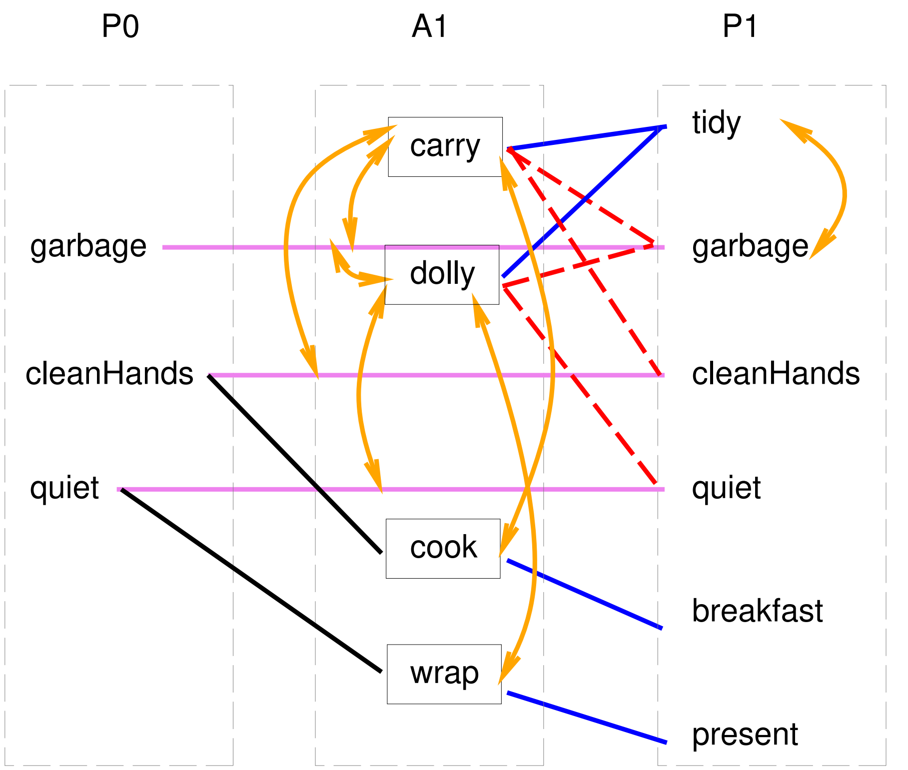
An action appear in A_{i+1} iff its preconditions appear & are mutex-free in P_i.



interference | inconsistence | inconsistent support | competing needs

Mutual exclusion: example

- carry & dolly are mutex with the maintenance action for garbage (interference+inconsistence)
- carry is mutex with the maintenance action for cleanHands (interference+inconsistence)
- dolly is mutex with the maintenance action for quiet (interference+inconsistence)
- carry and cook are mutex, dolly and wrap are mutex (interference)
- tidy is mutex with garbage (inconsistent support)



Graph formal definition (if it helps you)

The set of actions A includes maintenance actions m_p with
$\textsc{pre}(m_p) = \textsc{eff}^+(m_p) = \{p\}$

The graph alternates layers of propositions and actions $P_0, A_1, P_1, A_2, \ldots A_k, P_k$ and records mutex pairs μA_i and μP_i at each layer, such that:

- $P_0 = s_0$
- $\mu P_0 = \{\,\}$
- $A_{i+1} = \{a \in A \mid \textsc{pre}(a) \subseteq P_i$ and $\forall \{p, p'\} \in \mu P_i \; \{p, p'\} \not\subseteq \textsc{pre}(a)\}$
- $\mu A_{i+1} = \{\{a, a'\} \subseteq A_{i+1} \mid \textsc{eff}^-(a) \cap (\textsc{pre}(a') \cup \textsc{eff}^+(a')) \neq \{\,\}$ or
  $\exists \{p, p'\} \in \mu P_i$ s.t. $p \in \textsc{pre}(a)$ and $p' \in \textsc{pre}(a')\}$
- $P_{i+1} = \cup_{a \in A_{i+1}} \textsc{eff}^+(a)$
- $\mu P_{i+1} = \{\{p, p'\} \subseteq P_{i+1} \mid \forall a, a' \in A_{i+1}$ s.t. $p \in \textsc{eff}^+(a)$ and $p' \in \textsc{eff}^+(a')$,
  $\{a, a'\} \in \mu A_{i+1}\}$

Properties of the graph

Propositions and actions monotonically increase across levels;
Proposition and action mutexes monotonically decrease across levels:

$P_i \subseteq P_{i+1}$ and $A_i \subseteq A_{i+1}$
if $\{p, q\} \subseteq P_i$ and $\{p, q\} \notin \mu P_i$ then $\{p, q\} \notin \mu P_{i+1}$
if $\{a, b\} \subseteq A_i$ and $\{a, b\} \notin \mu A_i$ then $\{a, b\} \notin \mu A_{i+1}$

Proof: Each proposition $p \in P_{i+1}$ is supported by its maintenance action m_p. Two maintenance actions m_p and m_q are necessarily independent. If $\{p, q\} \subseteq P_i$ and if $\{p, q\} \notin \mu P_i$ then $\{m_p, m_q\} \notin \mu A_{i+1}$, hence $\{p, q\} \notin \mu P_{i+1}$. Similarly if $\{a, b\} \notin \mu A_i$ then they are independent and their preconditions in P_i are not mutex; these properties remain true at level $i + 1$.

The graph has a fixpoint n such that for all $i \geq n$:

$P_i = P_n$, $\mu P_i = \mu P_n$, $A_i = A_n$, and $\mu A_i = \mu A_n$

The size of the fixpoint graph is polynomial in that of the planning problem.

Usage of the graph

Necessary condition for plan existence:

If the goal propositions are present and mutex-free at some level P_k

$g \subseteq P_k$ and $\forall \{p, q\} \subseteq g \; \{p, q\} \notin \mu P_k$

then a k step parallel plan achieving the goal **might** exist

Heuristics for planning:

(may use the serial graph: any pair of actions at the same level are mutex)

- single proposition p: "cost" of achieving p is the index of the first level in which p appears
- set of propositions: max (or sum) of the individual costs, or index of the first level at which they all appear mutex-free

Planning:

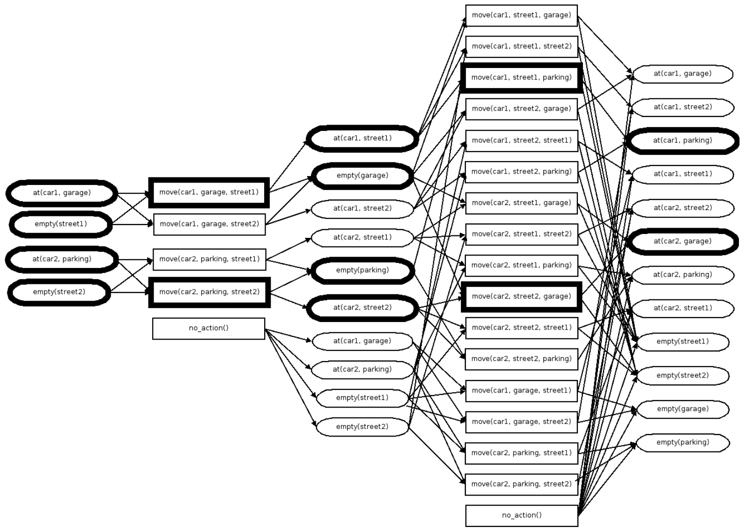
Graphplan algorithm: build the graph up until the necessary condition is reached; try extracting a plan from the graph, if this fails, extend the graph over one more level; repeat until success or termination condition (failure)

Plan extraction

Backward search, from the goal layer to the initial state layer.

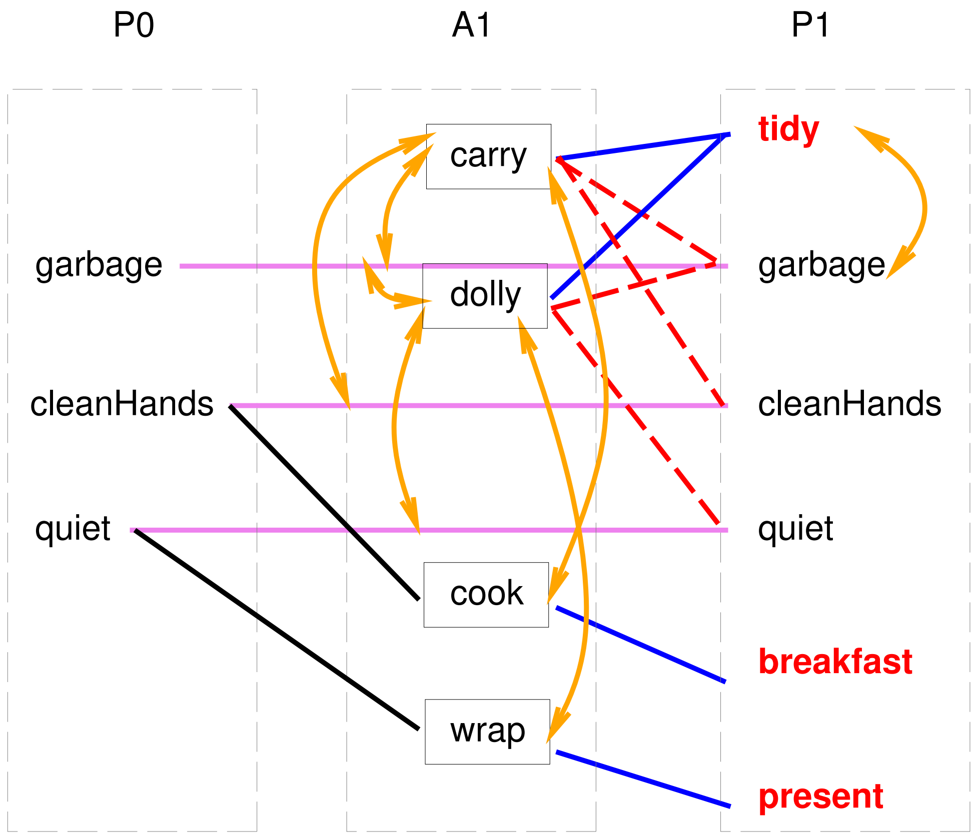
Works layer by layer:

- Select an open precondition at the current layer, and **choose** an action producing it. The action **must not be mutex** with any of the parallel actions already choosen for that layer.
- When there is no more open precondition at that layer, work on achieving, at the previous layer, the preconditions of the chosen actions.



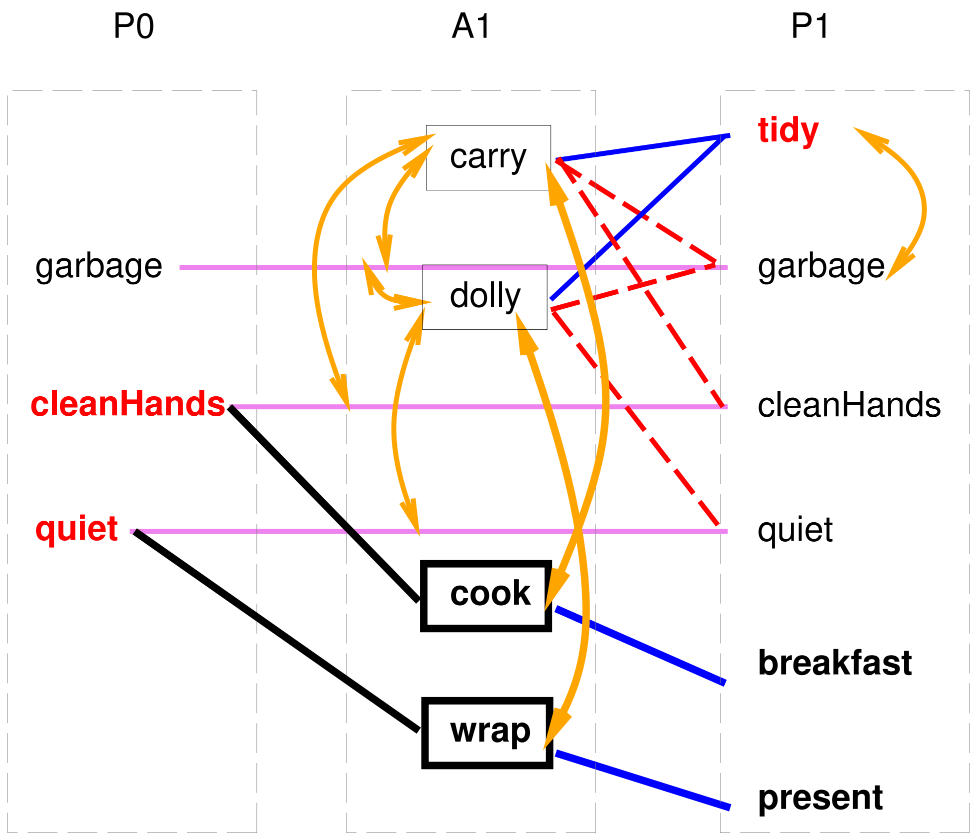
Example

The necessary condition for plan existence is satisfied at level 1 so we can attempt extraction.



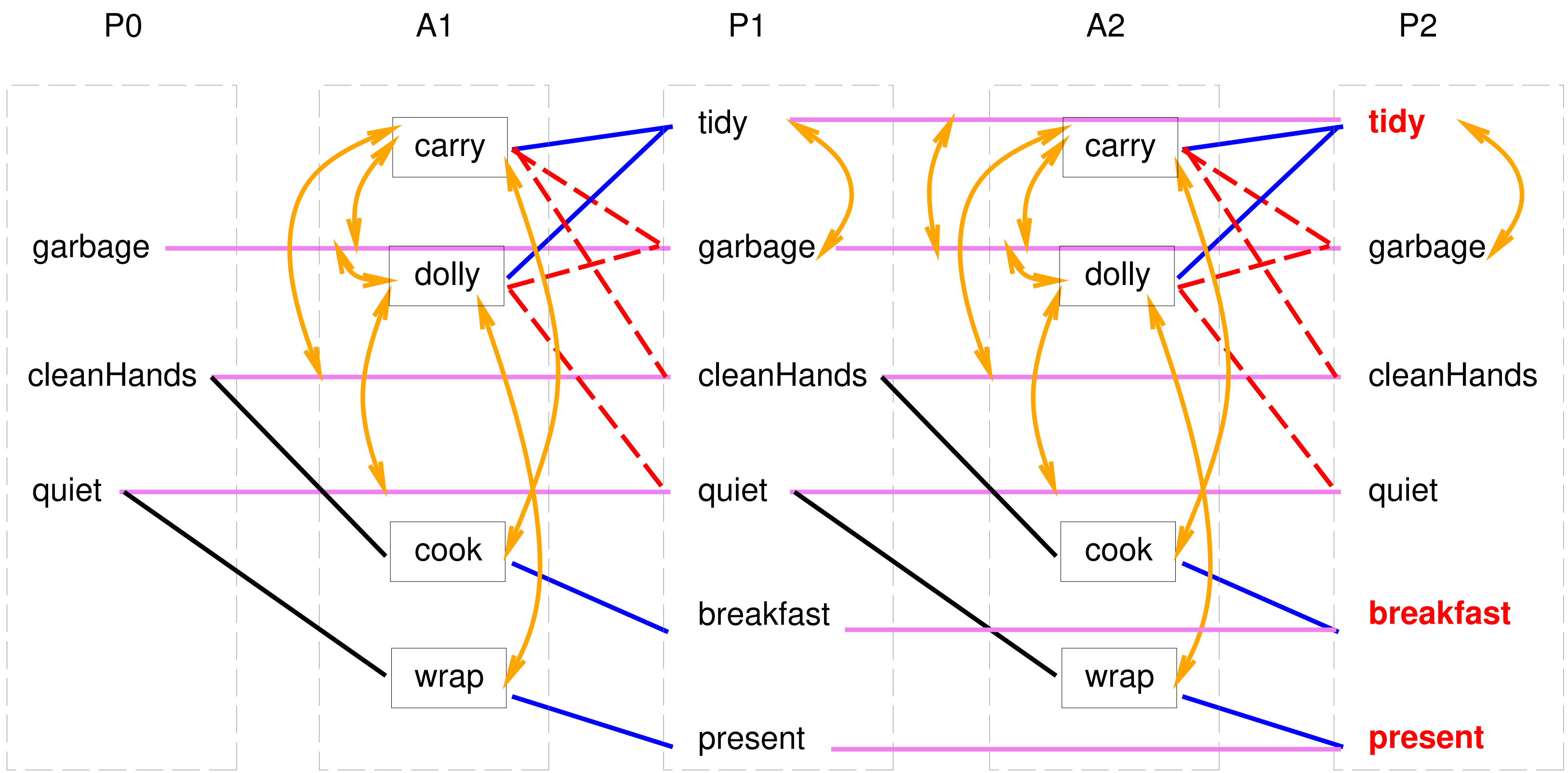
Example

At level 1, we can use cook to produce breakfast, wrap to produce present, but then we cannot achieve tidy because the actions producing it (carry and dolly) are mutex with either cook or wrap. So extraction fails.



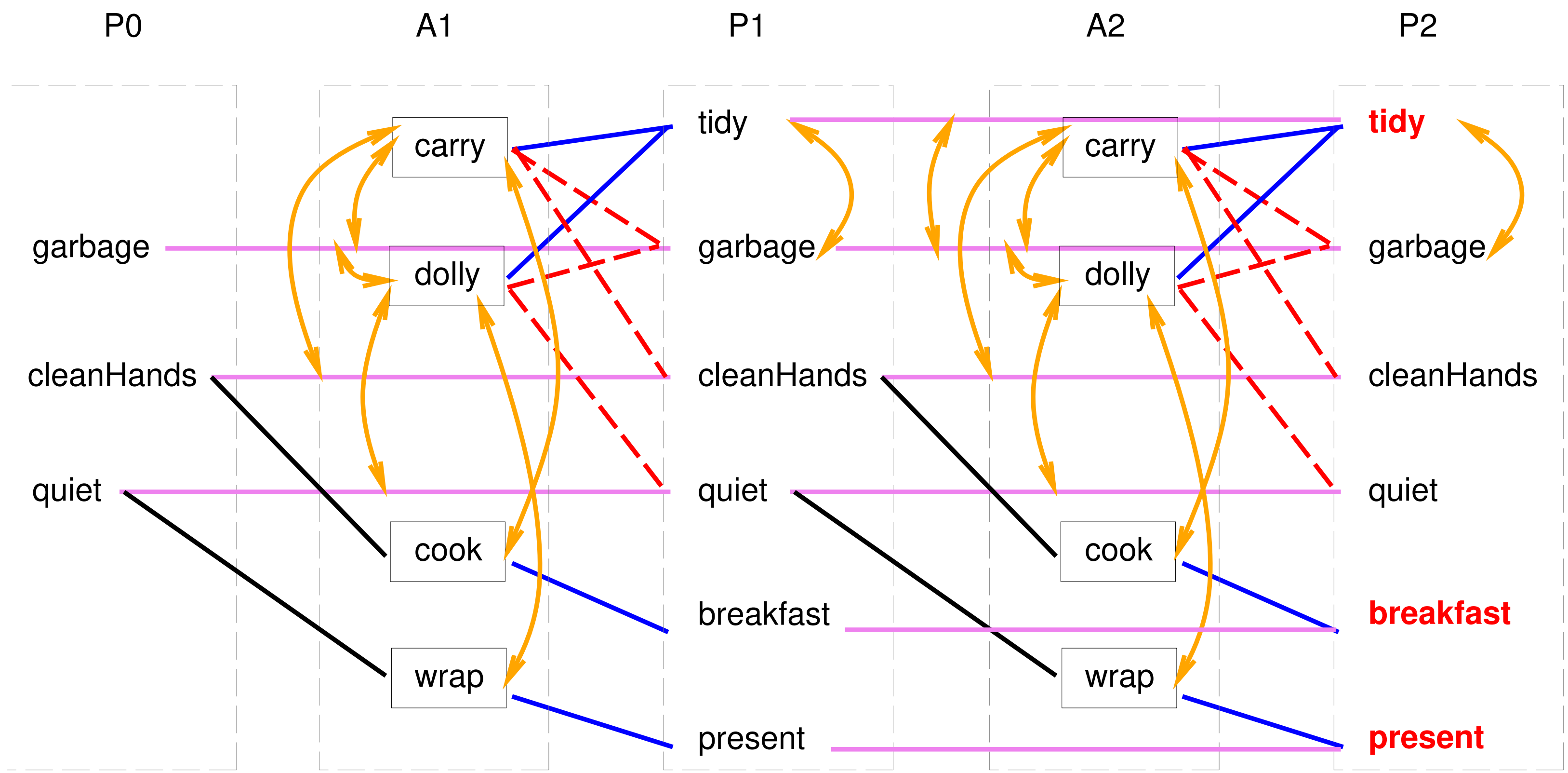
Example

And we extend the graph of one level. Note the apparition of new maintenance actions (for tidy, breakfast, and present), and of a new mutex between the tidy and garbage maitenance actions (competing needs).



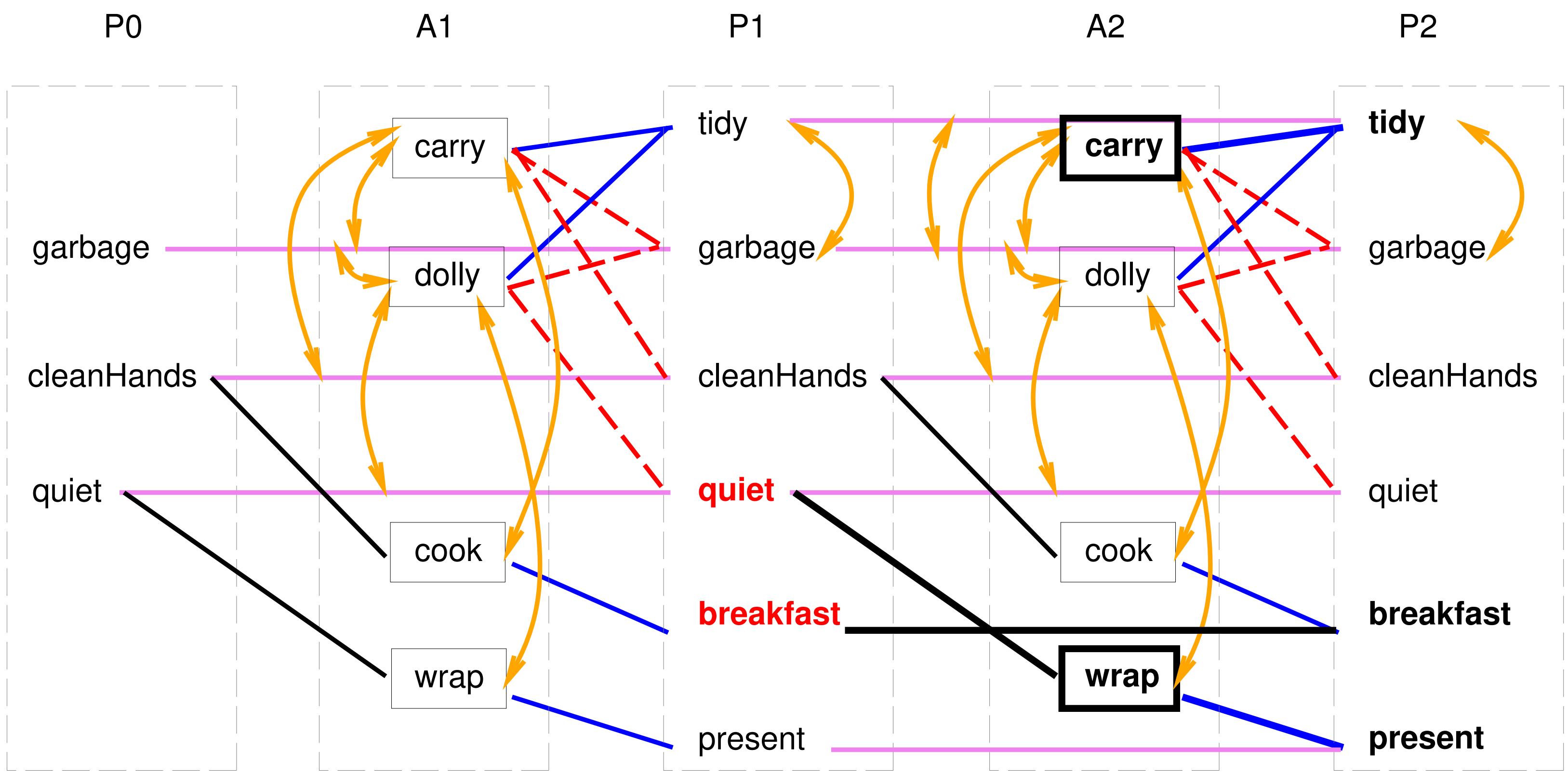
Example

There are 3 possibilities to achieve tidy (maintenance, carry, or dolly), 2 to achieve breakfast (maintenance or cook), and 2 to achieve present (maitenance or wrap)



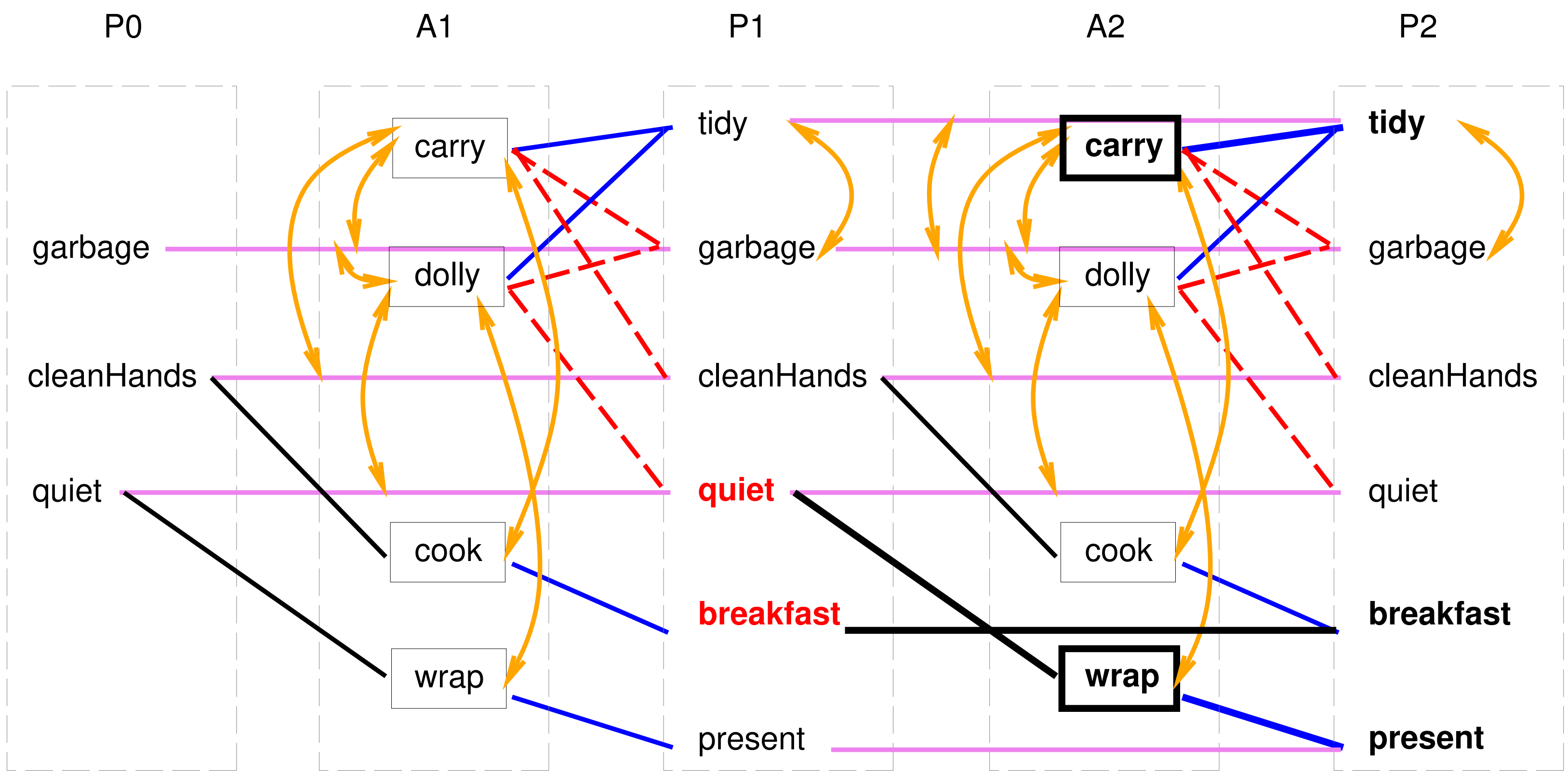
Example

There are 3 possibilities to achieve tidy (maintenance, carry, or dolly), 2 to achieve breakfast (maintenance or cook), and 2 to achieve present (maitenance or wrap), with several combinations being okay. One of them is carry, wrap, and maintenance of breakfast.



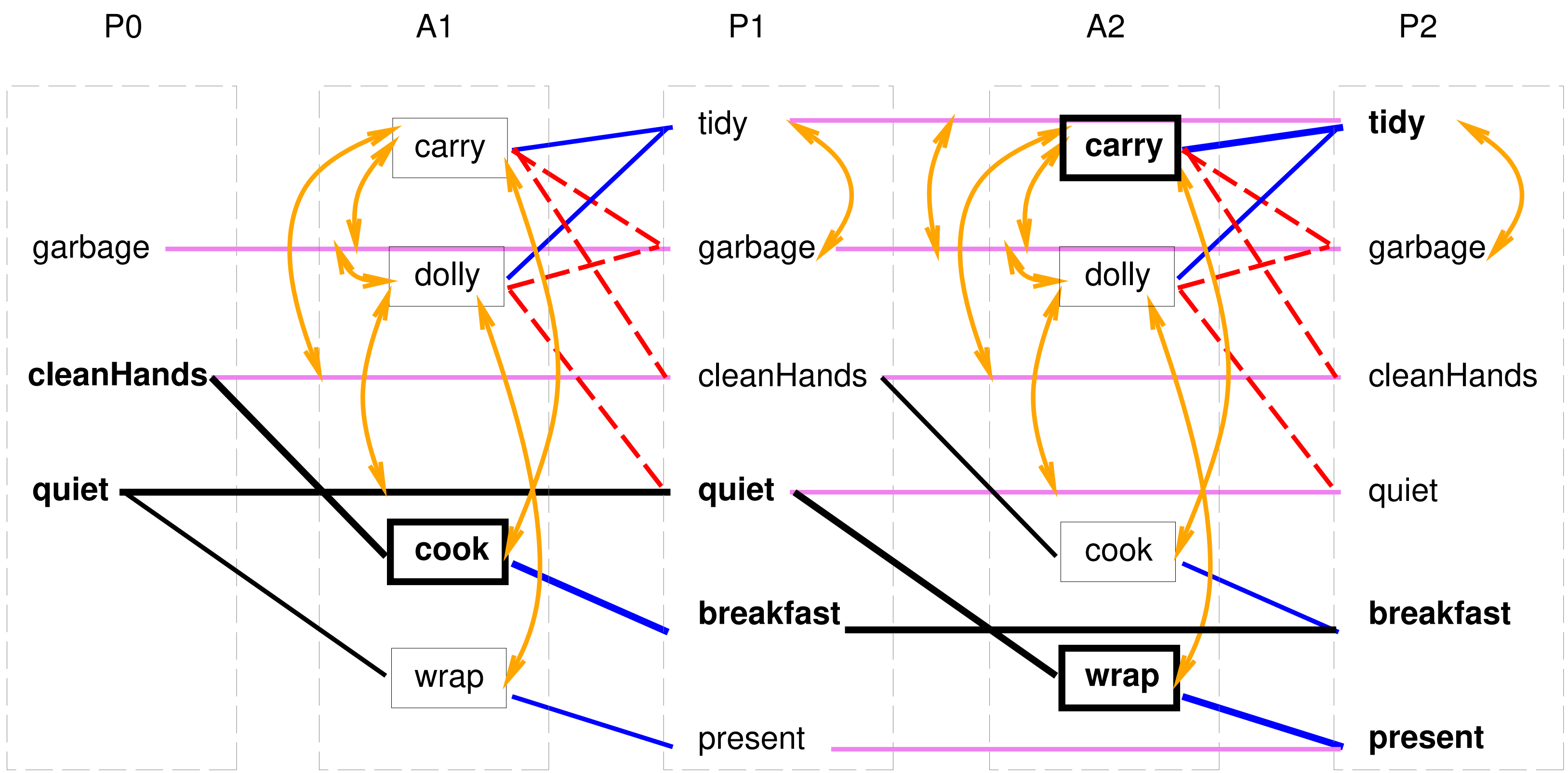
Example

There is only one possibility to achieve quiet and breakfast at level 1.



Example

There is only one possibility to achieve quiet and breakfast at level 1. This yields a solution whose parallel length is 2.



Plan extraction algorithm

function $\textsc{Extract}(i, g_i, \pi_i)$ **returns** a parallel plan, or failure

    if $i = 0$ **then return** $\langle\rangle$
    if $g_i \neq \{\,\}$ **then**
        select any $p \in g_i$
        $E \leftarrow \{a \in A_i \mid p \in \textsc{eff}^+(a)$ and $\forall\, b \in \pi_i\ \{a, b\} \notin \mu\mathbf{A}_i\}$
        if $E = \{\,\}$ **then return** failure
        choose $a \in E$
        return $\textsc{Extract}(i, g_i \setminus \textsc{eff}^+(a), \pi_i \cup \{a\})$
    else
        $\pi \leftarrow \textsc{Extract}(i-1, \cup_{a \in \pi_i} \textsc{Pre}(a), \{\,\})$
        if $\pi =$ failure **then return** failure
        return $\pi.\pi_i$
    end

call: $\textsc{Extract}(k, g, \{\,\})$ where k is the last layer in the graph

Heuristics: pick p with highest level cost, a with smallest precondition cost.

Properties of Graphplan

Graphplan is sound. Is Graphplan complete?

When can it terminate asserting failure?

in the worst case

- stop when $k > |S|$: complete but inefficient
- stop when $P, A, \mu A, \mu P$ reach a fix point: incomplete unless PSPACE = NP

Record nogoods: speeds up termination whilst ensuring completeness

- Δ_i records inconsistent proposition sets (nogoods) at level i
- when $\text{EXTRACT}(i, g_i, \{\})$ fails, add g_i to Δ_i
- when $g_i \supseteq \delta$ and $\delta \in \Delta_i$, $\text{EXTRACT}(i, g_i, \{\})$ returns failure
- nogoods monotonically decrease
- stop when $P, A, \mu A, \mu P, \Delta$ reach a fix point

The graph has a fixpoint n such that for all $i \geq n$: $P_i = P_n$, $\mu P_i = \mu P_n$, $A_i = A_n$, and $\mu A_i = \mu A_n$

Size of the fixpoint graph polynomial in that of the planning problem. Plan extraction NP-complete.

Planning via satisfiability testing

We can use a SAT solver to achieve the same results as Graphplan, without the algorithmic complications

Idea introduced in the SATPLAN planner [Kautz & Selman, KR 1992]

SAT solvers have become very efficient; unit propagation and clause learning are powerful; SAT planning is very efficient when plans are short

Unlike other approaches, choices are non-directional

Expressive! Easy to add control knowledge

Reminder on SAT

From the KR lectures:

- a SAT problem consists of
  - a set of boolean variables V
  - a set (conjunction) of clauses C (disjunctions of literals)

A solution (model, satisfying assignment) is a valuation (true, false) for each of the variables in V that make all clauses in C true

Satisfiable formula:

$$(v_1 \vee \neg v_2) \wedge (\neg v_1 \vee v_2 \vee v_3) \wedge \neg v_1$$

$v_2 \Rightarrow F$ T or F T $\Rightarrow$ $v_1 = F$

Solution:

$$\begin{aligned} v_1 &= \text{FALSE} \\ v_2 &= \text{FALSE} \\ v_3 &= \text{FALSE}(or\,\text{TRUE}) \end{aligned}$$

Principles of SAT planning

How can SAT planning work?

- planning is PSPACE-complete and SAT is NP-complete!
- SAT can only solve the **bounded** plan existence problem: does there exists a plan with k (or less) steps?
- encode this question as a SAT problem $\Phi_k = (V_k, C_k)$
- if Φ_k is satisfiable, the solution yields a plan
- solving the original planning problem requires solving a **sequence** of SAT problems, e.g. increment k until a plan is found

So SAT is only a bounded/partial planning finder.

SAT encodings of planning: variables

Let $\langle P, A, s_0, g\rangle$ be a (grounded) STRIPS planning problem, where P is the set of ground atoms, A is the set of ground actions, $s_0 \subseteq P$ is the initial state and $g \subseteq P$ is the goal

The SAT variables V_k are:

- $p@t$ for each $p \in P$ and $t \in \{0, \ldots, k\}$
  $p@t$ is a fluent denoting that p holds at time step t
  e.g. on(A, B)@3
- $a@t$ for each $a \in A$ and $t \in \{0, \ldots, k-1\}$
  $a@t$ is an action fluent denoting that a occurs at time step t
  e.g. stack(R1, A, B)@2

We can build literals from these variables, e.g. ¬on(A, B)@0, ¬stack(R1, A, B)@1.

SAT encodings of planning: clauses

Initial state and goal:

- All propositions that are true in the initial state must hold at time step 0 and only those (closed world assumption). All goal propositions must hold at time step k (open world assumption)

$$\bigwedge_{p \in s_0} p@0 \wedge \bigwedge_{p \notin s_0} \neg p@0 \wedge \bigwedge_{p \in g} p@k$$

Action preconditions and effects:

- For each action occuring at time step t then its preconditions must be true at time step t, its positive effects are true at time step $t+1$ and its negative effects are false at time step $t+1$. For each $a \in A$ and each $t \in \{0, \ldots, k-1\}$:

$$a@t \Rightarrow \Big(\bigwedge_{p \in \mathrm{PRE}(a)} p@t \wedge \bigwedge_{p \in \mathrm{EFF}^+(a)} p@t{+}1 \wedge \bigwedge_{p \in \mathrm{EFF}^-(a)} \neg p@t{+}1 \Big)$$

SAT encodings of planning: clauses

Frame Problem: need for logical planning encodings to formalise and reason about the value of propositions that are not modified by actions.

The STRIPS formalism avoids the frame problem.

Explanatory frame axioms:

- The only way a fluent can change is via the occurrence of an action that changes it. For each $p \in P$ and each $t \in \{0, \ldots, k-1\}$:

$$(\neg p@t \wedge p@t{+}1 \Rightarrow \bigvee_{\substack{a \in A \\ p \in \mathbf{EFF}^{+}(a)}} a@t) \wedge (p@t \wedge \neg p@t{+}1 \Rightarrow \bigvee_{\substack{a \in A \\ p \in \mathbf{EFF}^{-}(a)}} a@t)$$

Mutual exclusion axioms:

- Action pairs that interfere cannot occur in parallel. For each $\{a, a'\} \subseteq A$ such that $\textsc{pre}(a) \cap \textsc{eff}^{-}(a') \neq \{\,\}$, and each $t \in \{0, \ldots, k-1\}$:

$$\neg a@t \vee \neg a'@t$$

Example

Operator	Precondition	Effect
cook()	{cleanHands}	{breakfast}
wrap()	{quiet}	{present}
carry()	{}	{tidy, ¬garbage, ¬cleanHands}
dolly()	{}	{tidy, ¬garbage, ¬quiet}

$s_0 = \{\text{garbage, cleanHands, quiet}\}$
$g = \{\text{breakfast, present, tidy}\}$
$k = 1$

initial state / goal

$cleanHands@0 \wedge garbarge@0 \wedge quiet@0 \wedge$
$\neg breakfast@0 \wedge \neg present@0 \wedge \neg tidy@0 \wedge$
$breakfast@1 \wedge present@1 \wedge tidy@1$

actions preconditions / effects

$cook@0 \Rightarrow cleanHands@0 \wedge breakfast@1$
$wrap@0 \Rightarrow quiet@0 \wedge present@1$
$carry@0 \Rightarrow tidy@1 \wedge \neg garbage@1 \wedge \neg cleanHands@1$
$dolly@0 \Rightarrow tidy@1 \wedge \neg garbage@1 \wedge \neg quiet@1$

mutual exclusion

$\neg cook@0 \vee \neg carry@0$
$\neg wrap@0 \vee \neg dolly@0$

frame axioms

$\neg breakfast@0 \wedge breakfast@1 \Rightarrow cook@0$
$breakfast@0 \wedge \neg breakfast@1 \Rightarrow False$
$\neg cleanHands@0 \wedge cleanHands1 \Rightarrow False$
$cleanHands@0 \wedge \neg cleanHands1 \Rightarrow carry@0$
$\neg garbage@0 \wedge garbage@1 \Rightarrow False$
$garbage@0 \wedge \neg garbage@1 \Rightarrow carry@0 \vee dolly@0$
$\neg present@0 \wedge present@1 \Rightarrow wrap@0$
$present@0 \wedge \neg present@1 \Rightarrow False$
$\neg quiet@0 \wedge quiet@1 \Rightarrow False$
$quiet@0 \wedge \neg quiet@1 \Rightarrow dolly@0$
$\neg tidy@0 \wedge tidy@1 \Rightarrow carry@0 \vee dolly@0$
$tidy@0 \wedge \neg tidy@1 \Rightarrow False$

Example (clausal form)

initial state / goal

$cleanHands@0$
$garbarge@0$
$quiet@0$
$\neg breakfast@0$
$\neg present@0$
$\neg tidy@0$
$breakfast@1$
$present@1$
$tidy@1$

actions preconditions / effects

$\neg cook@0 \vee cleanHands@0$
$\neg cook@0 \vee breakfast@1$
$\neg wrap@0 \vee quiet@0$
$\neg wrap@0 \vee present@1$
$\neg carry@0 \vee tidy@1$
$\neg carry@0 \vee \neg garbage@1$
$\neg carry@0 \vee \neg cleanHands@1$
$\neg dolly@0 \vee tidy@1$
$\neg dolly@0 \vee \neg garbage@1$
$\neg dolly@0 \vee \neg quiet@1$

mutual exclusion

$\neg cook@0 \vee \neg carry@0$
$\neg wrap@0 \vee \neg dolly@0$

frame axioms

$breakfast@0 \vee \neg breakfast@1 \vee cook@0$
$\neg breakfast@0 \vee breakfast@1$
$cleanHands@0 \vee \neg cleanHands1$
$\neg cleanHands@0 \vee cleanHands1 \vee carry@0$
$garbage@0 \vee \neg garbage@1$
$\neg garbage@0 \vee garbage@1 \vee carry@0 \vee dolly@0$
$present@0 \vee \neg present@1 \vee wrap@0$
$\neg present@0 \vee present@1$
$quiet@0 \vee \neg quiet@1$
$\neg quiet@0 \vee quiet@1 \vee dolly@0$
$tidy@0 \vee \neg tidy@1 \vee carry@0 \vee dolly@0$
$\neg tidy@0 \vee tidy@1$

Well-known encoding improvements

Fluent mutexes:

- Use the planning graph to get them

  $\neg$holding(R1, A)@t $\vee$ $\neg$handempty(R1)@t

Search control knowledge:

- very easy with SAT

  If a package is at its goal location, then it must remain there

  at(p, l)@t $\wedge$ goal_loaction$(p, l) \Rightarrow at(p, l)$@$t + 1$

Parallel plans:

- alternative semantics: allow actions $\{a_1 \ldots a_n\}$ at time step t if they can be executed in **at least one** order; can substantially reduce number of plan steps [Rintanen et. al, Aus AI 2007]

Well-known encoding improvements

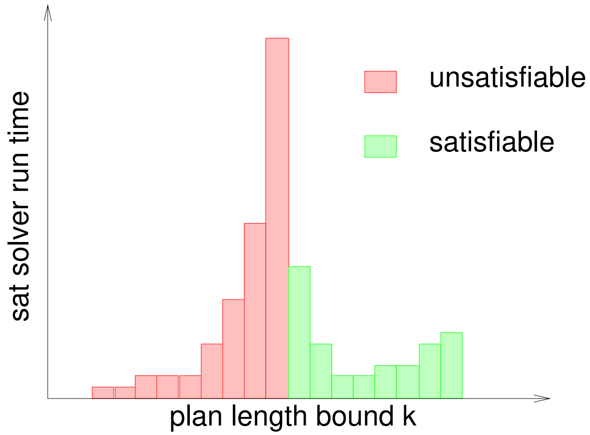
Operator splitting and axiom factoring [Ernst et. al, IJCAI 1997]:

- n-ary action fluent split into conjunction of n unary action fluents
- stack(R1, A, B)@t replaced by stack1(R1)@t $\wedge$ stack2(A)@t $\wedge$ stack3(B)@t
- reduces number of action fluents dramatically
- factoring substitutes only parts of the conjunction into axioms: only includes the part of the action conjunction that is relevant to a given fluent

$$\text{stack1(R1)@}t \Rightarrow \text{handempty(R1)@}t{+}1$$
$$\text{stack1(R1)@}t \wedge \text{stack2(A)} \Rightarrow \text{holding(R1, A)@}t \wedge \neg\text{holding(R1, A)@}t{+}1$$
$$\text{stack3(B)@}t \Rightarrow \text{clear(B)@}t \wedge \text{clear(B)@}t{+}1$$
$$\text{stack2(A)@}t \wedge \text{stack3(B)@}t \Rightarrow \text{on(A, B)@}t{+}1$$

- avoids CNF conversion blowup

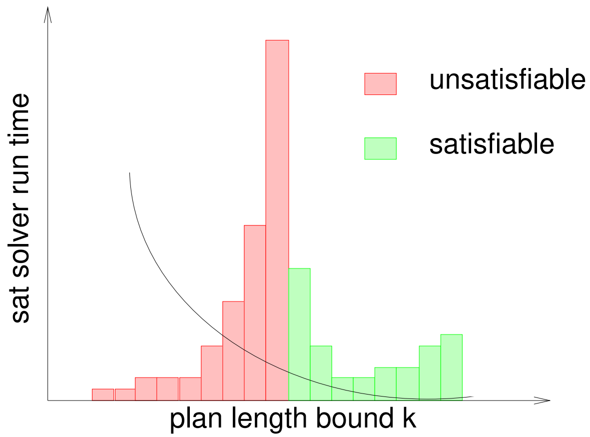
Evaluation strategies



Which Φ_k problems to solve, with which computation time?

- increasing nb steps: $k = l, l+1, l+2, \ldots$ with l lower bound
- decreasing nb steps: $k = b, b-1, b-2, \ldots$ with b upper bound
- dychotomy and other sequential strategies to find a plan with good performance ratio [Streeter & Smith, ICAPS 2007]
- run in parallel with geometric run-time γ^k, $\gamma < 1$ to find arbitrary plan [Rintanen, ECAI 2004]; move frontier when a Φ_k is proven unsat.

Evaluation strategies



Which Φ_k problems to solve, with which computation time?

- increasing nb steps: $k = l, l+1, l+2, \ldots$ with l lower bound
- decreasing nb steps: $k = b, b-1, b-2, \ldots$ with b upper bound
- dychotomy and other sequential strategies to find a plan with good performance ratio [Streeter & Smith, ICAPS 2007]
- run in parallel with geometric run-time γ^k, $\gamma < 1$ to find arbitrary plan [Rintanen, ECAI 2004]; move frontier when a Φ_k is proven unsat.

Summary

Graph-based planning produces parallel plans. A planning graph is a relaxation of the state space, which gives us a necessary condition for the existence of a parallel plan of a given length. If one really exist, it can be extracted by backward search through the graph.

SAT planning uses a SAT solver to solve the bounded (parallel) plan generation problem. Logical planning formalisms must deal with the frame problem.

Plan-space planning produces partially-ordered plans. This approach does not commit to orderings or bindings unless necessary. It searches the space of partial plans, refining the plan at each step to remove flaws.

Current planning research extends these methods to handle time, uncertainty, multiple agents, hybdrid (discrete-continuous) systems, and many other aspects of real world applications.