Planning-Graph and Satisfiability Techniques

Chapter 10

Outline

◇ Planning graph techniques: May 15 ← subtle

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

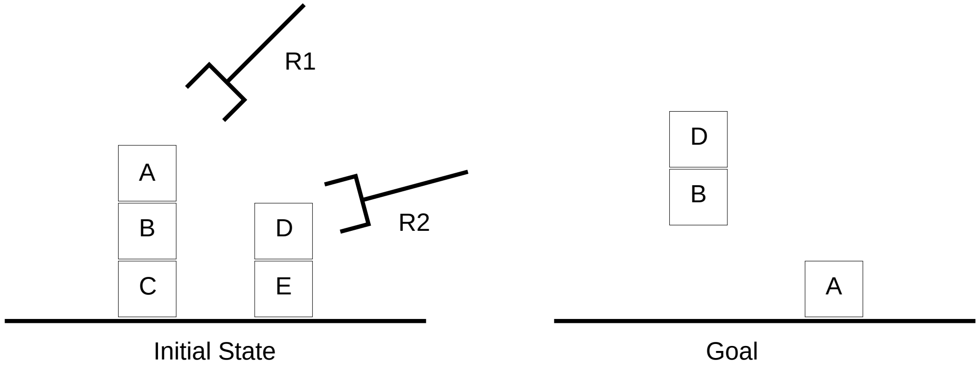
◇ SAT planning techniques: ← much easier

- 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.

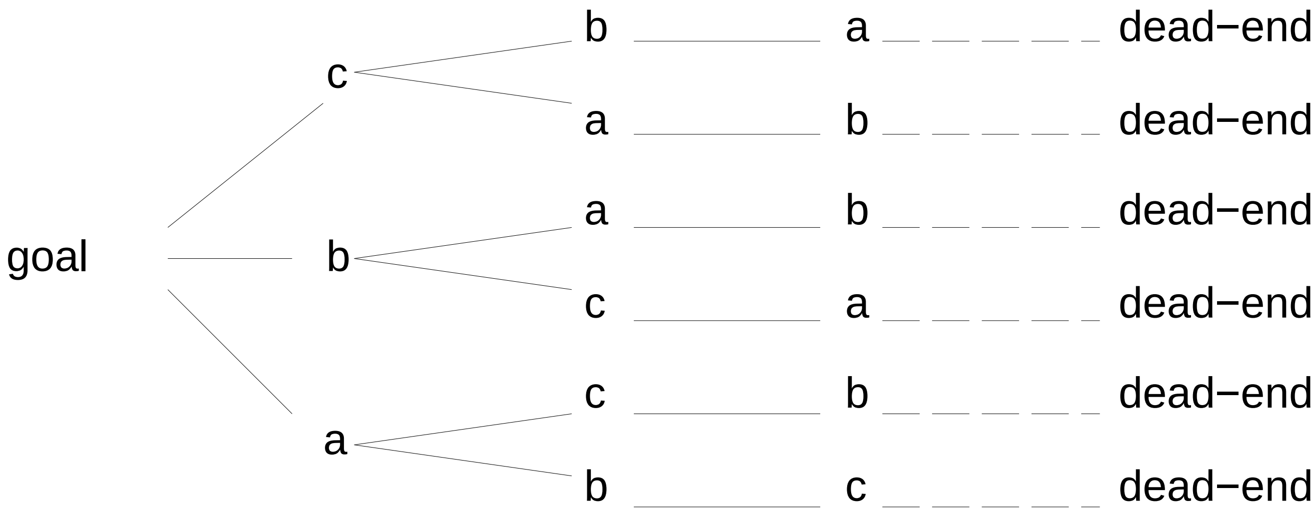


sequence:
$\langle \text{unstack}(R1, A, B), \text{unstack}(R2, D, E), \text{putdown}(R1, A), \text{stack}(R2, D, B) \rangle$

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:



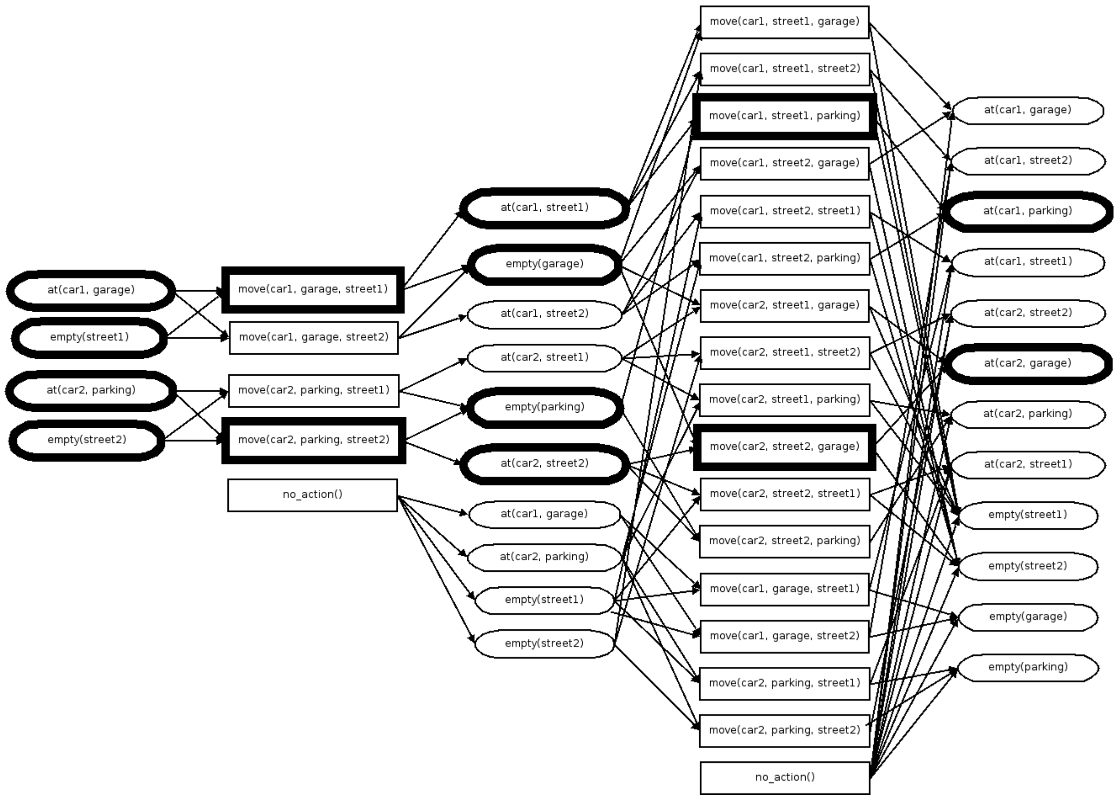
Not ordering actions that can take place in parallel can speed up planning

Motivation

State space search wastes time exploring individual states in all their details

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

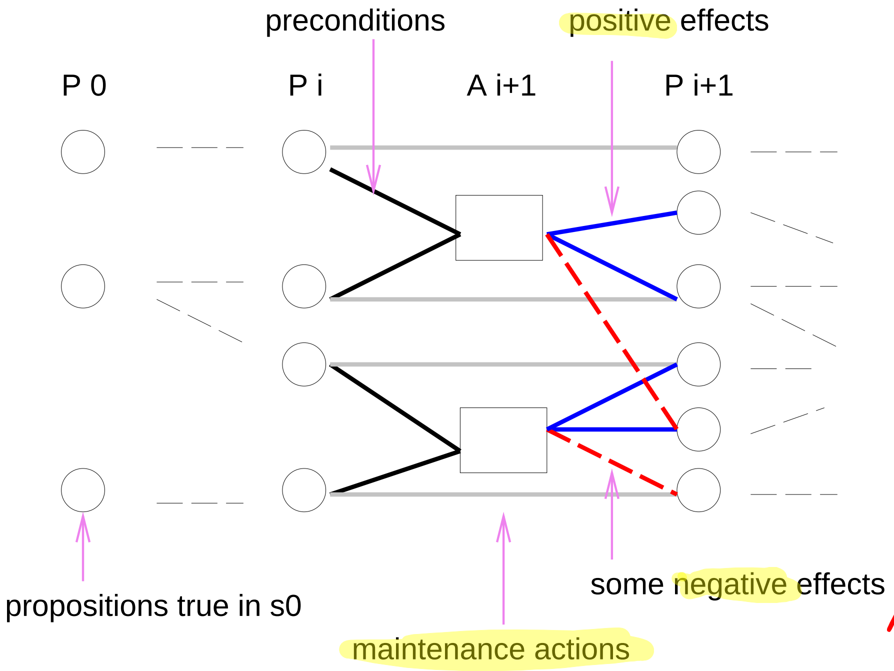
This is how Graphplan [Blum & First, IJCAI 1995] implements this idea:

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}



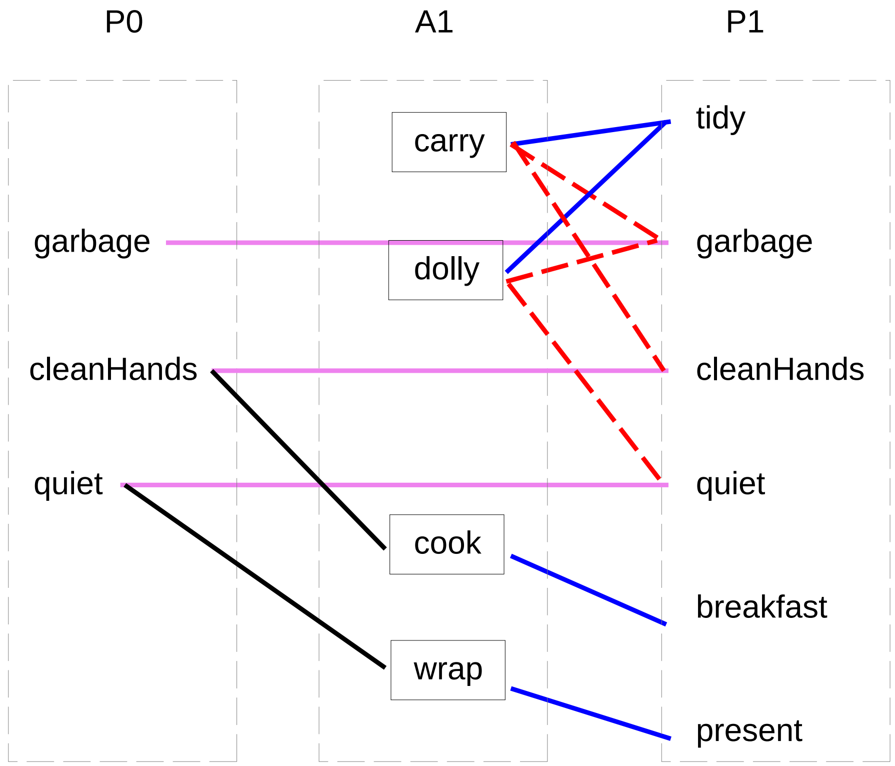
// threats, not negative effects in the usual manner

The planning graph: example

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

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

$s_0 = \{$garbage, cleanHands, quiet$\}$
$g = \{$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)

Mutual exclusion

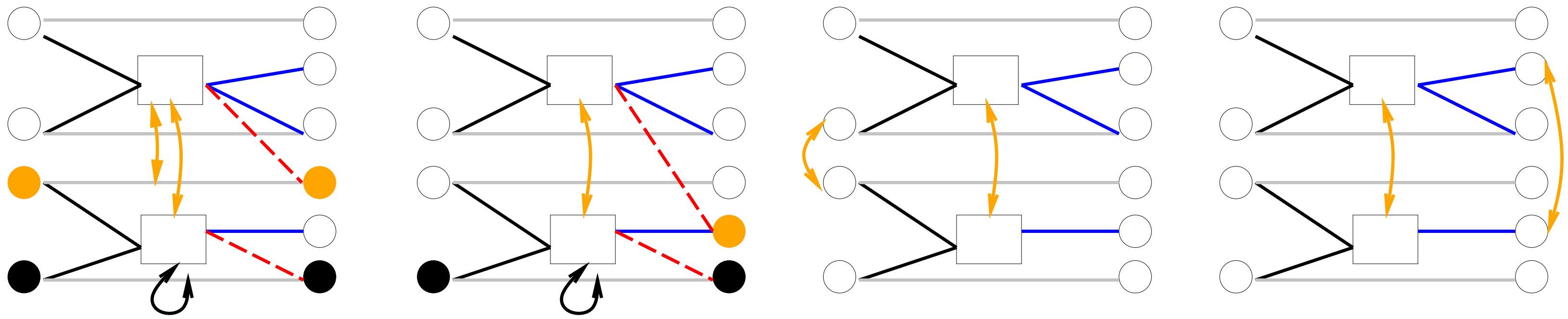
Two actions at the same level of the graph are mutex if they:

- 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

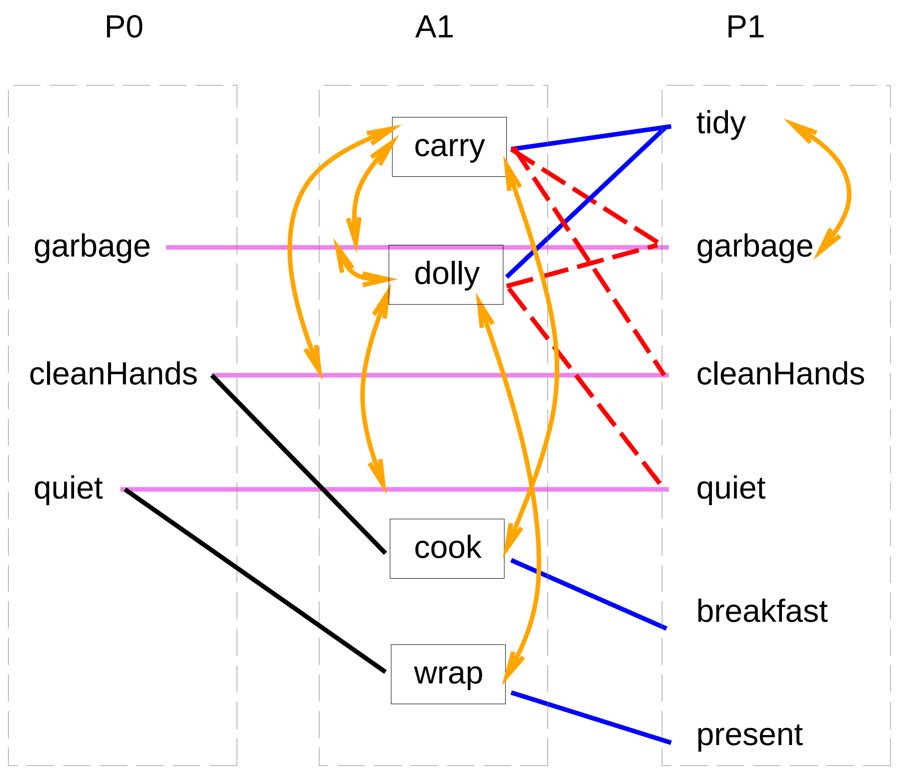
An action appear in A_{i+1} iff its preconditions appear & are mutex-free in P_i.



interference inconsistence competing needs inconsistent support

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
$\text{PRE}(m_p) = \text{EFF}^+(m_p) = \{p\}$

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:

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

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

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

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

NOT Guaranteed

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:

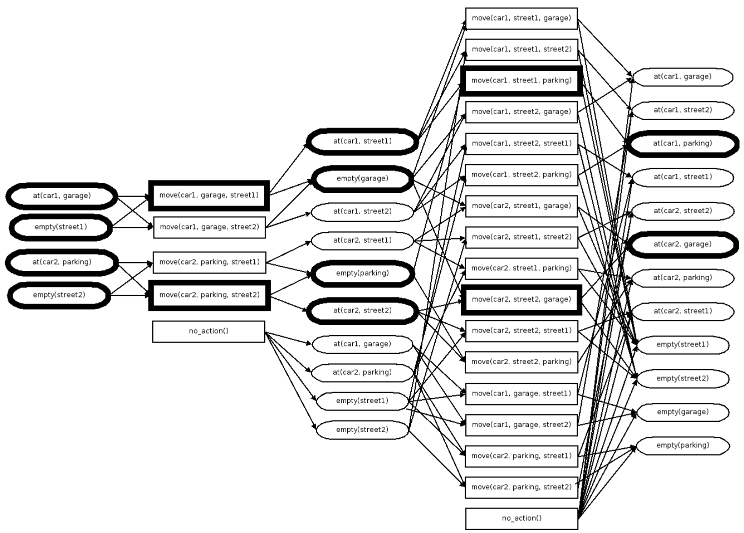
Graphplan algorithm: build the graph up until the necessary condition is reached; try extracting a plan from the graph, if this fails, extend the graph over one more level; repeat until success or termination condition (failure)

Plan extraction

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

Works layer by layer:

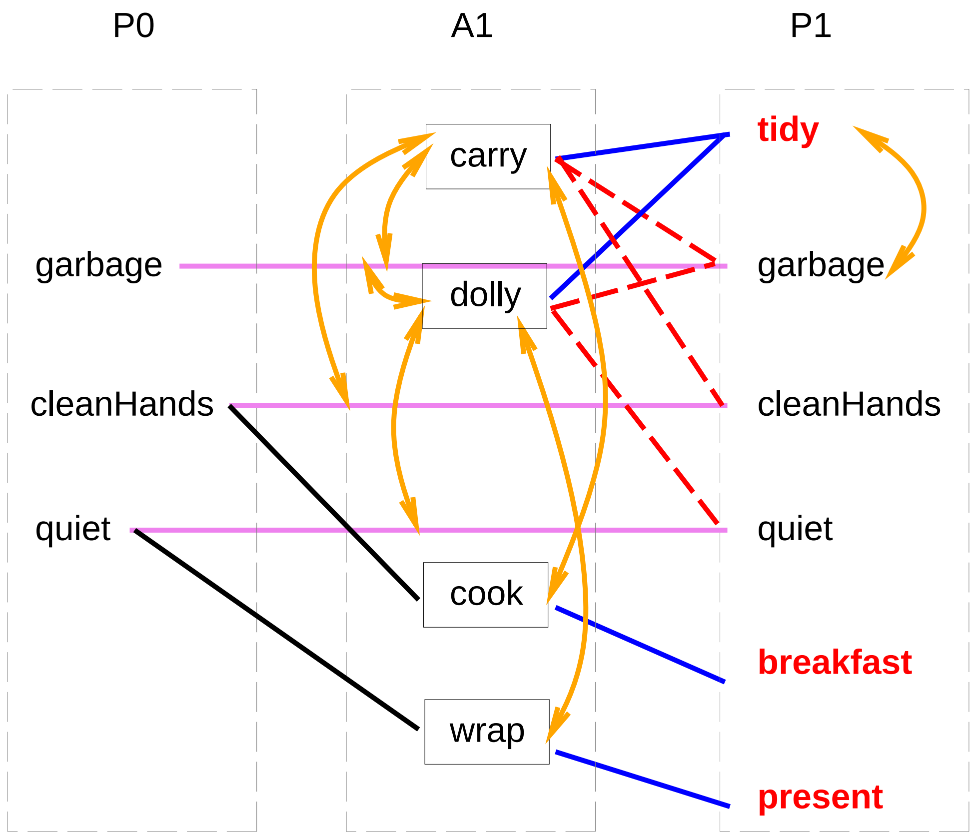
- Select an open precondition at the current layer, and **choose** an action producing it. The action **must not be mutex** with any of the parallel actions already choosen for that layer.
- When there is no more open precondition at that layer, work on achieving, at the previous layer, the preconditions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



Direction

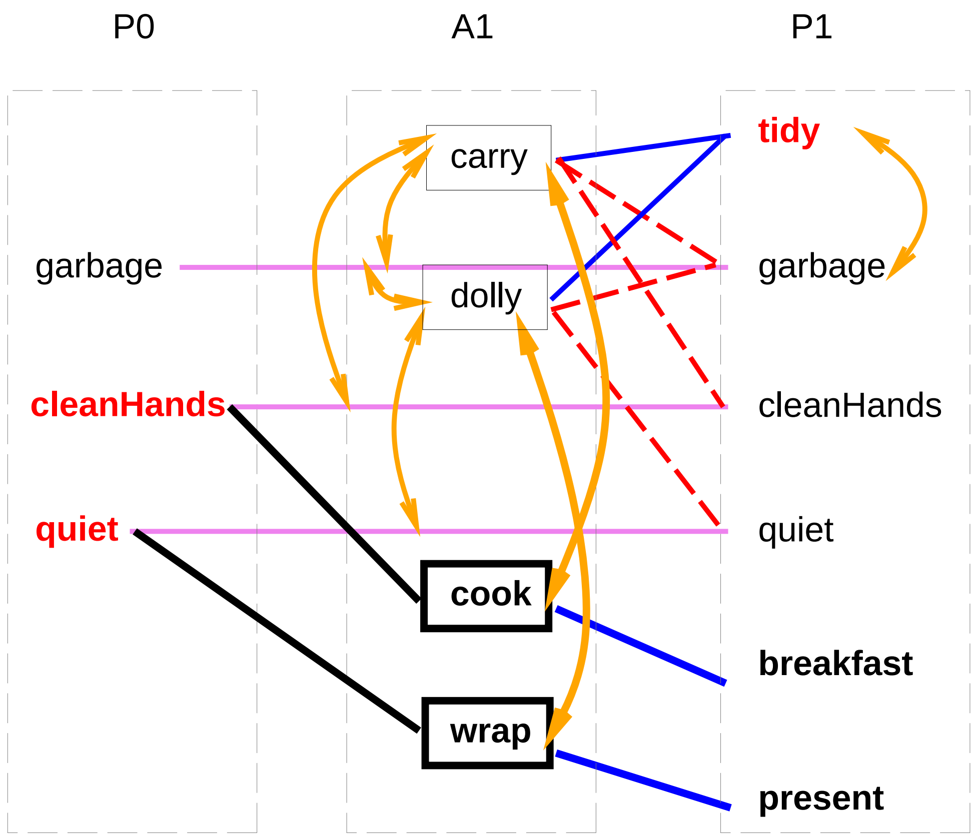
Example

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



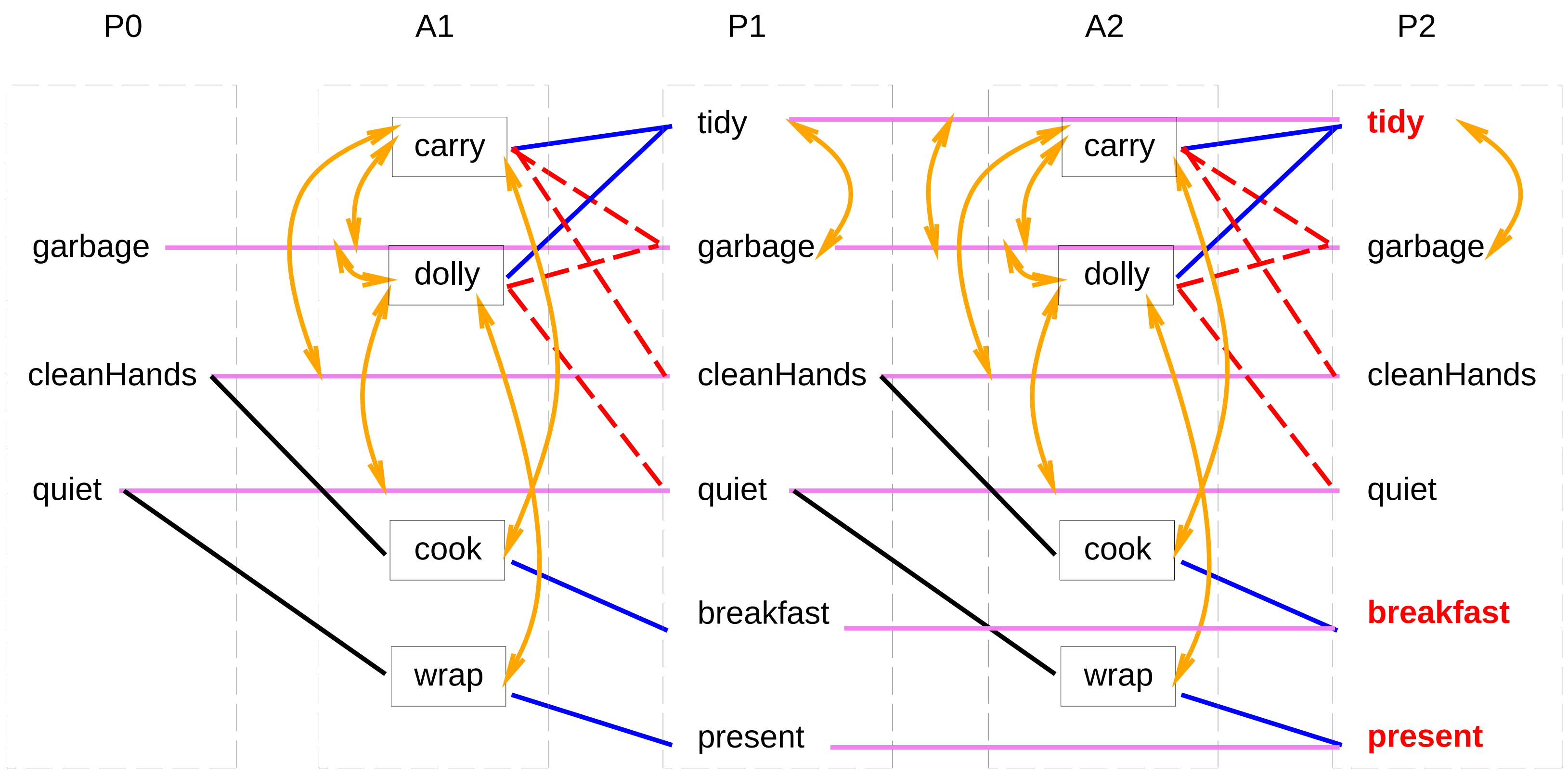
Example

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.



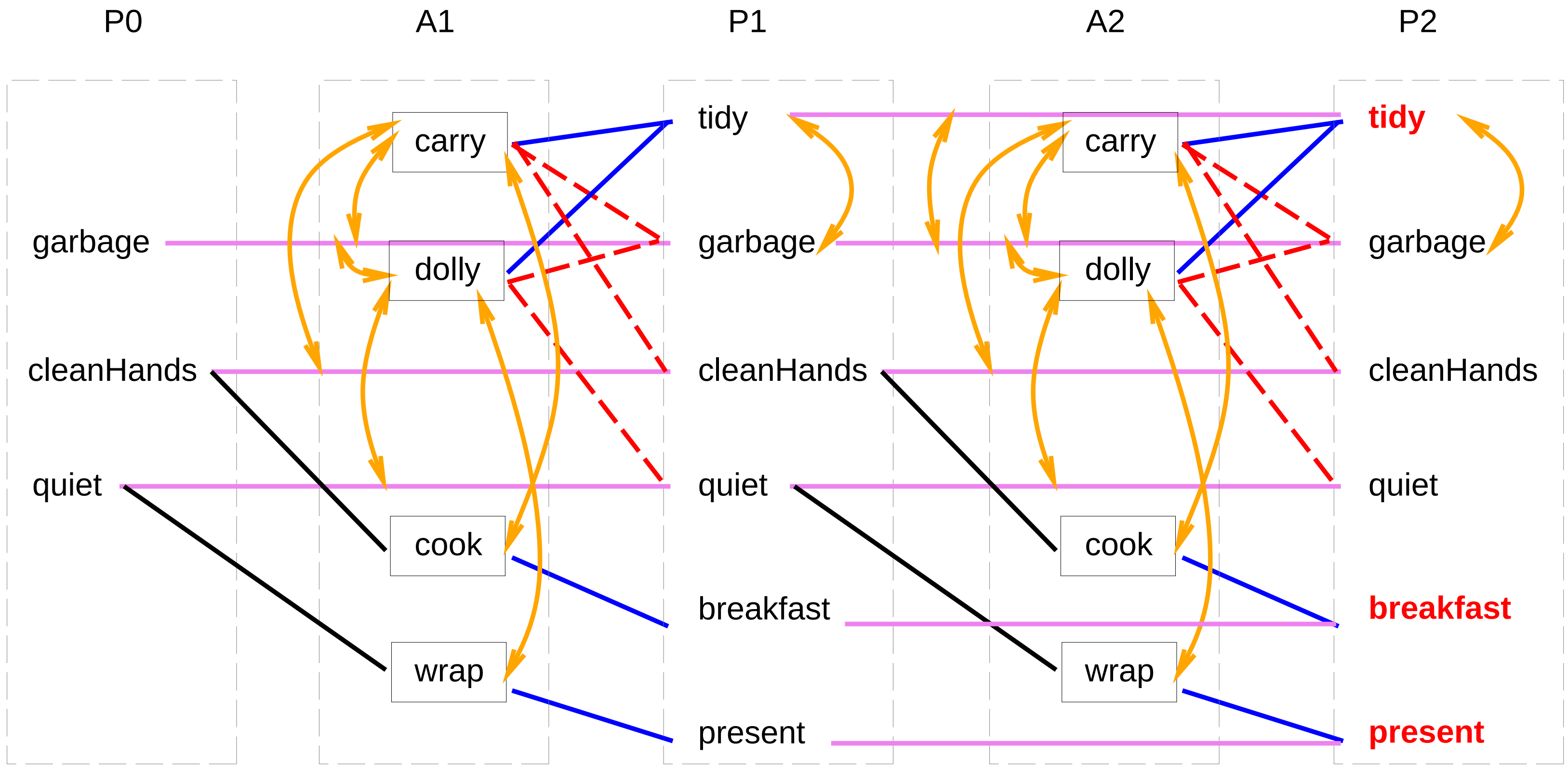
Example

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).



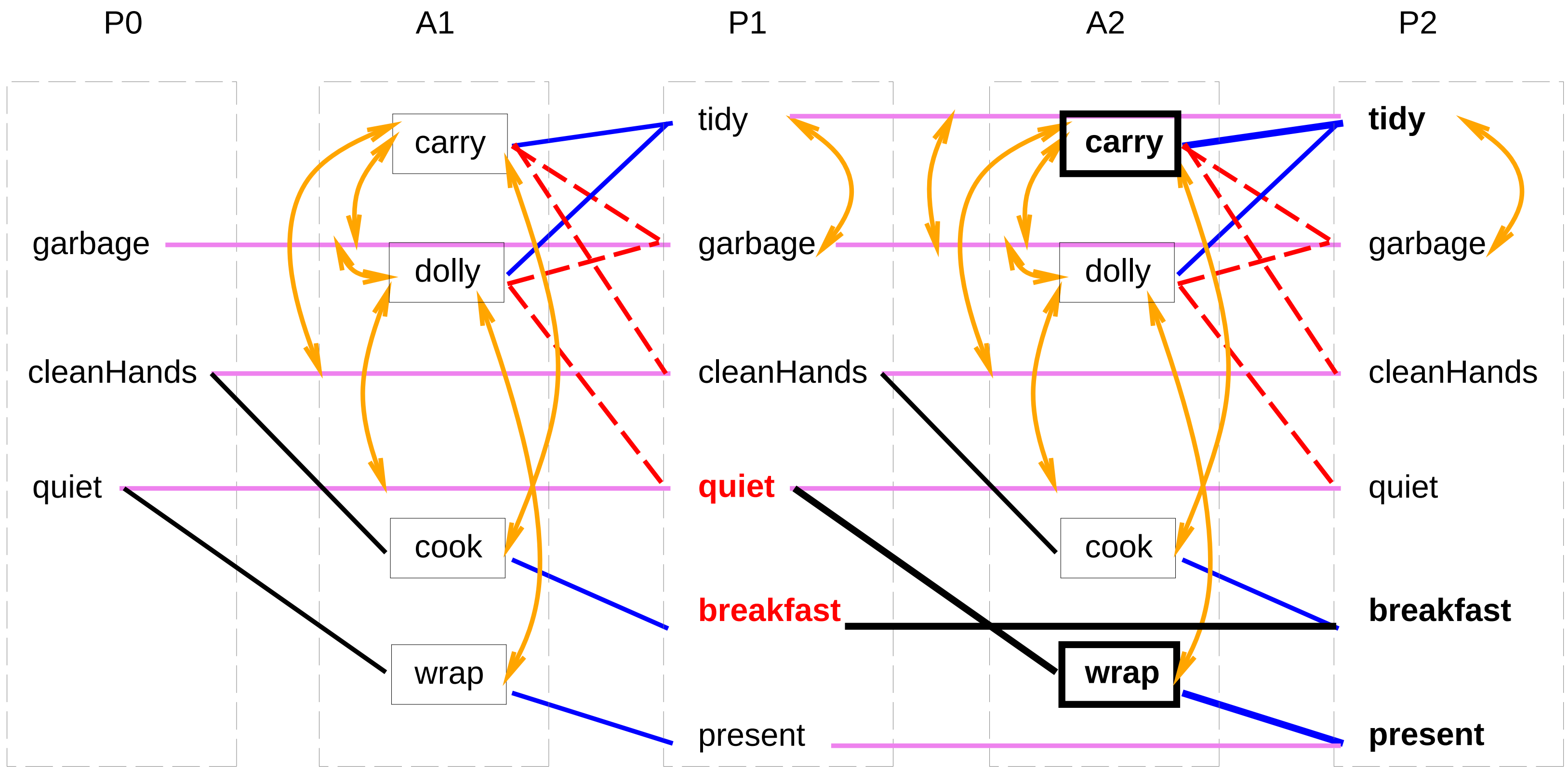
Example

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)



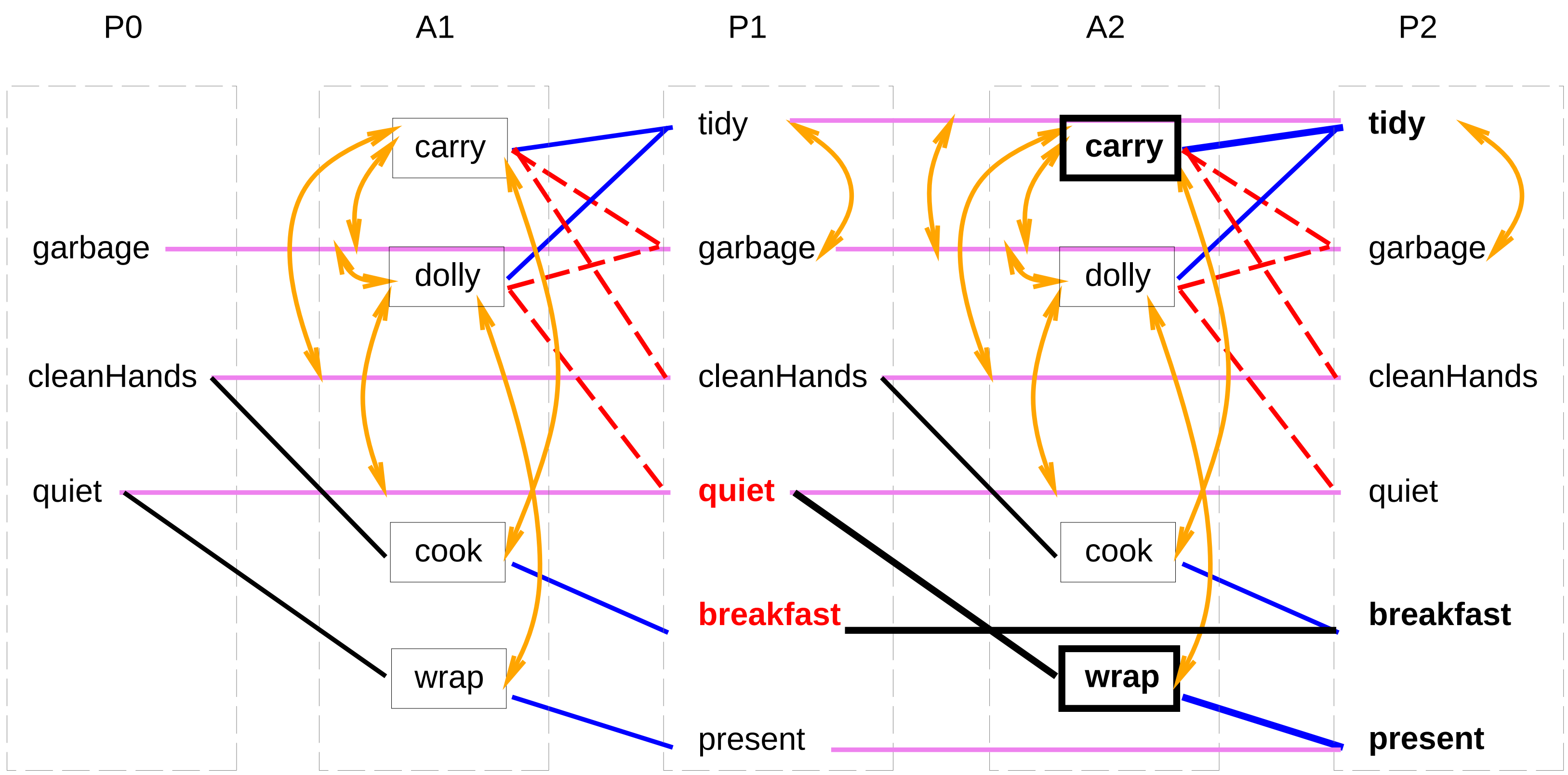
Example

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.



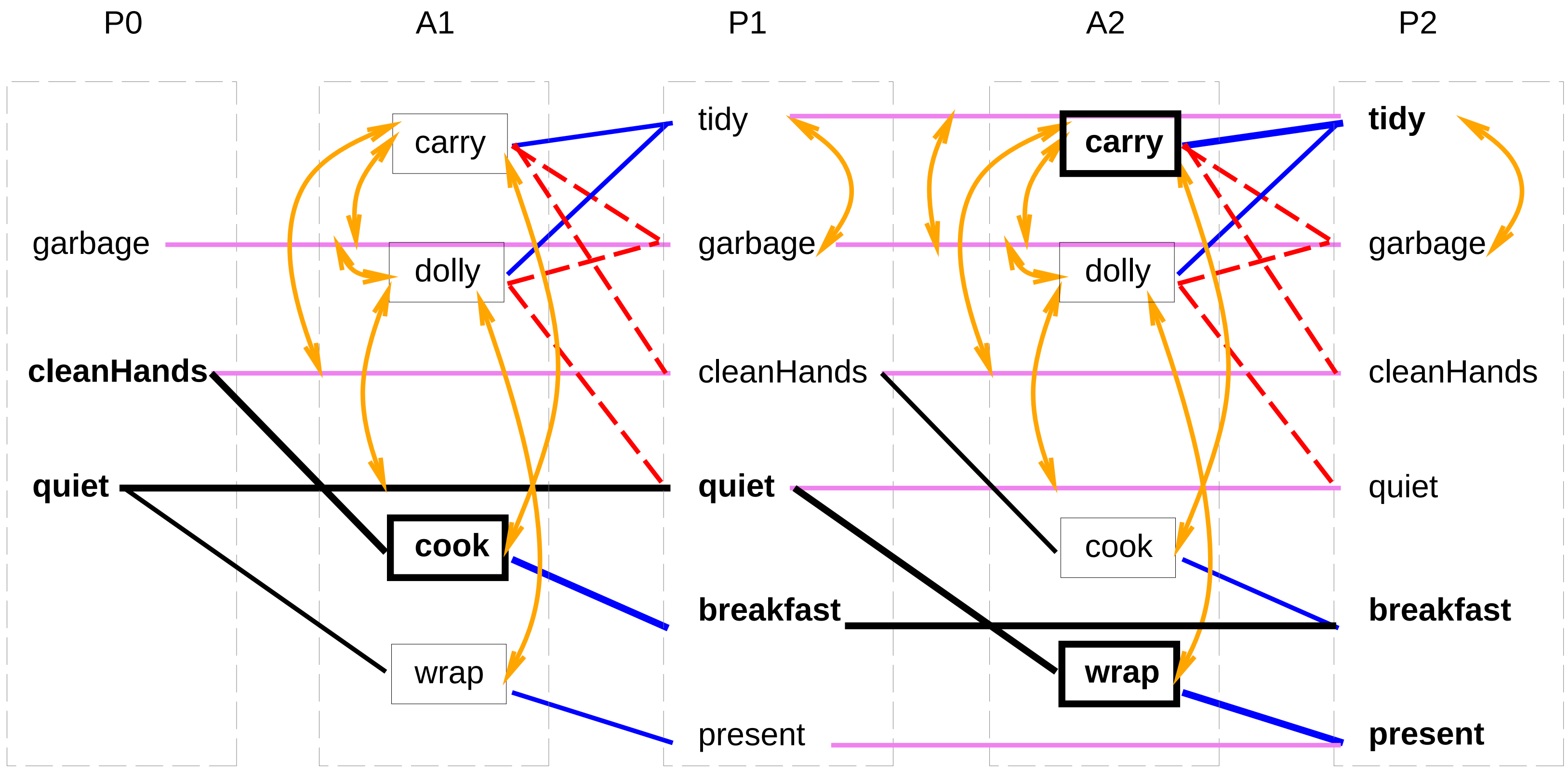
Example

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 EXTRACT(i, g_i, π_i) **returns** a parallel plan, or failure

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

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

Meaning ?

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

$P_i \leftarrow i$ minimum

$a \leftarrow$ req. min No. of preconditions

Properties of Graphplan

Graphplan is sound. Is Graphplan complete?

When can it terminate asserting failure?

- 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 $\textsc{Extract}(i, g_i, \pi_i)$ fails, add g_i to Δ_i
- when $g_i \supseteq \delta$ and $\delta \in \Delta_i$, $\textsc{Extract}(i, g_i, \pi_i)$ 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.