Automated Planning

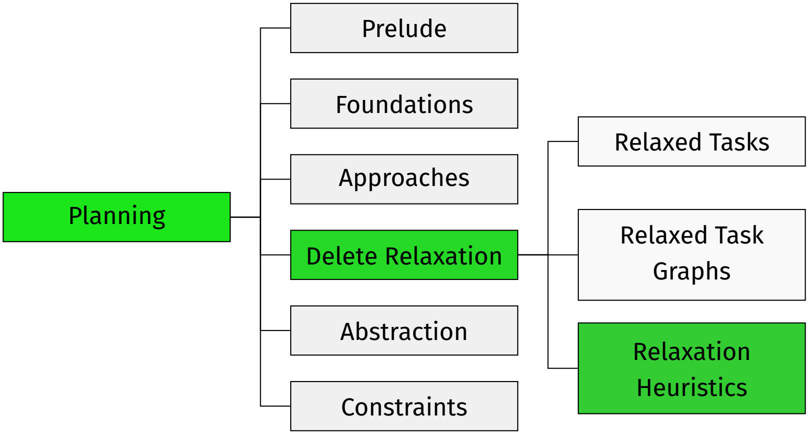
D4. Delete Relaxation: h^{max} and h^{add}

Jendrik Seipp

Linköping University

based on slides from the AI group at the University of Basel

Content of this Course



Introduction

Delete Relaxation Heuristics

- In this chapter, we introduce heuristics based on delete relaxation.
- Their basic idea is to propagate information in relaxed task graphs, similar to the previous chapter.
- Unlike the previous chapter, we do not just propagate information about whether a given node is reachable, but estimates how expensive it is to reach the node.

Reminder: Running Example

We will use the same running example as in the previous chapter:

$\Pi = \langle V, I, \{o_1, o_2, o_3, o_4\}, \gamma\rangle$ with

$$
\begin{aligned}
V &= \{a, b, c, d, e, f, g, h\} \\
I &= \{a \mapsto \mathbf{T}, b \mapsto \mathbf{T}, c \mapsto \mathbf{F}, d \mapsto \mathbf{T}, \\
&\quad\ e \mapsto \mathbf{F}, f \mapsto \mathbf{F}, g \mapsto \mathbf{F}, h \mapsto \mathbf{F}\} \\
o_1 &= \langle c \lor (a \land b), c \land ((c \land d) \rhd e), 1\rangle \\
o_2 &= \langle \top, f, 2\rangle \\
o_3 &= \langle f, g, 1\rangle \\
o_4 &= \langle f, h, 1\rangle \\
\gamma &= e \land (g \land h)
\end{aligned}
$$

h^{max} and h^{add}

Associating Costs with RTG Nodes

Basic intuitions for associating costs with RTG nodes:

- To apply an operator, we must pay its cost.
- To make an OR node true, it is sufficient to make one of its predecessors true.
  - ⇝ Therefore, we estimate the cost of an OR node as the minimum of the costs of its predecessors.
- To make an AND node true, all its predecessors must be made true first.
  - ⇝ We can be optimistic and estimate the cost as the maximum of the predecessor node costs.
  - ⇝ Or we can be pessimistic and estimate the cost as the sum of the predecessor node costs.

h^{max} Algorithm

(Differences to reachability analysis algorithm highlighted.)

Computing h^{max} Values

Associate a *cost* attribute with each node.

for all nodes n:

$n.cost := \infty$

while no fixed point is reached:

Choose a node n.

if n is an AND node that is not an effect node:

$n.cost := \max_{n' \in predecessors(n)} n'.cost$

if n is an effect node for operator o:

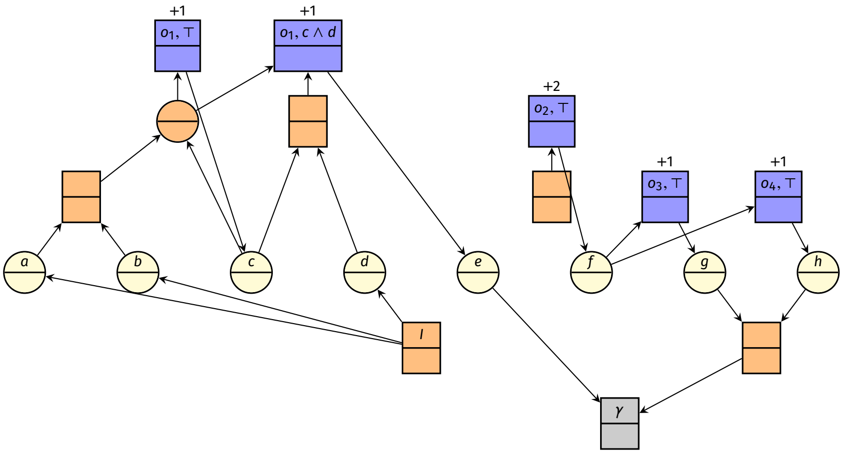
$n.cost := cost(o) + \max_{n' \in predecessors(n)} n'.cost$

if n is an OR node:

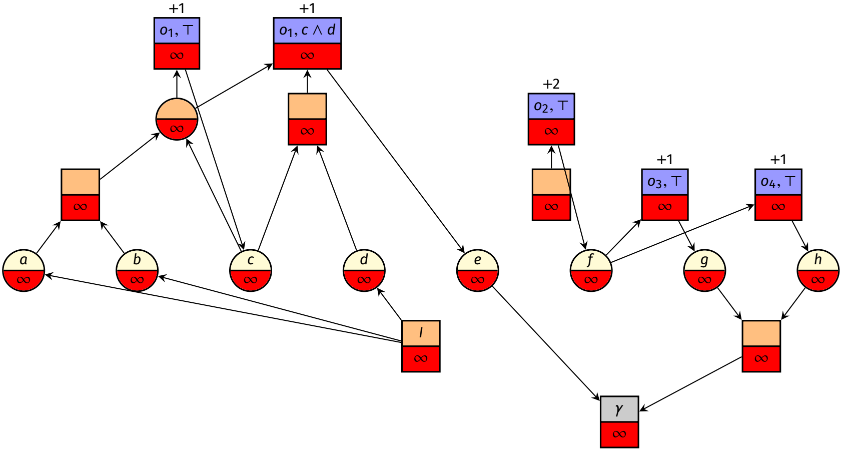
$n.cost := \min_{n' \in predecessors(n)} n'.cost$

The overall heuristic value is the cost of the goal node, $n_\gamma.cost$.

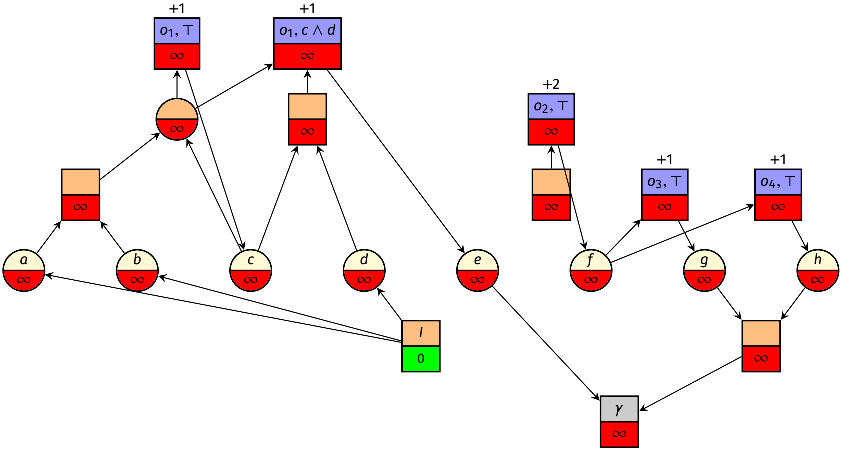
$h^{\max}$: Example



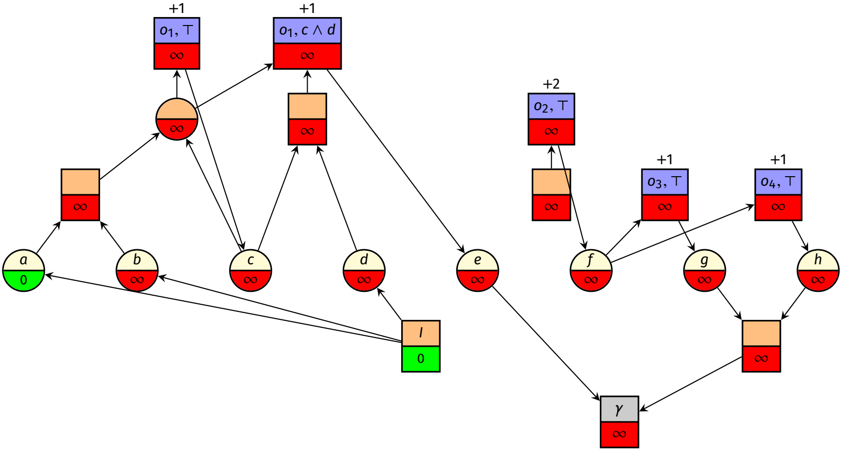
h^{max}: Example



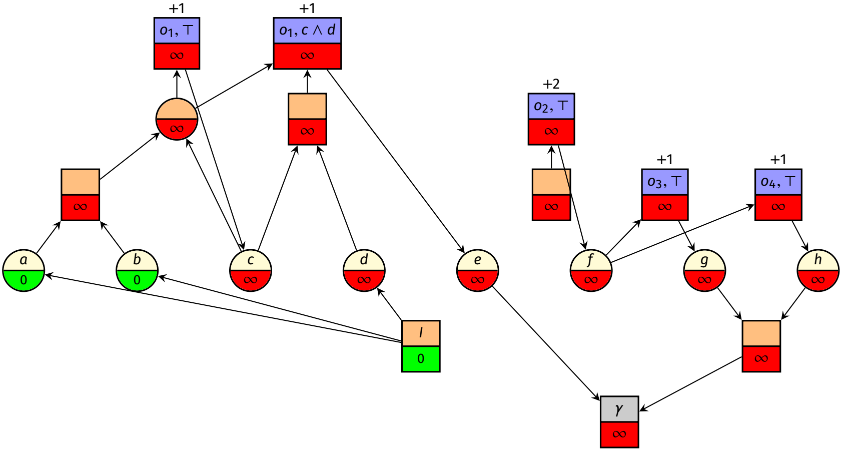
h^{max}: Example



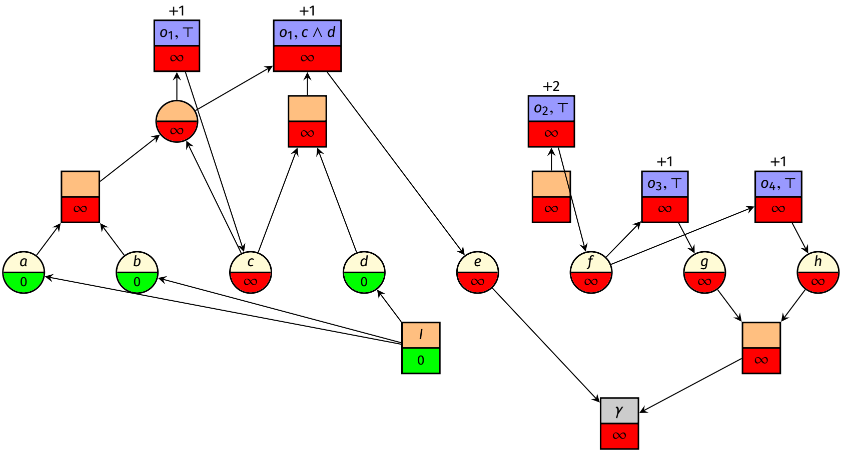
h^{max}: Example



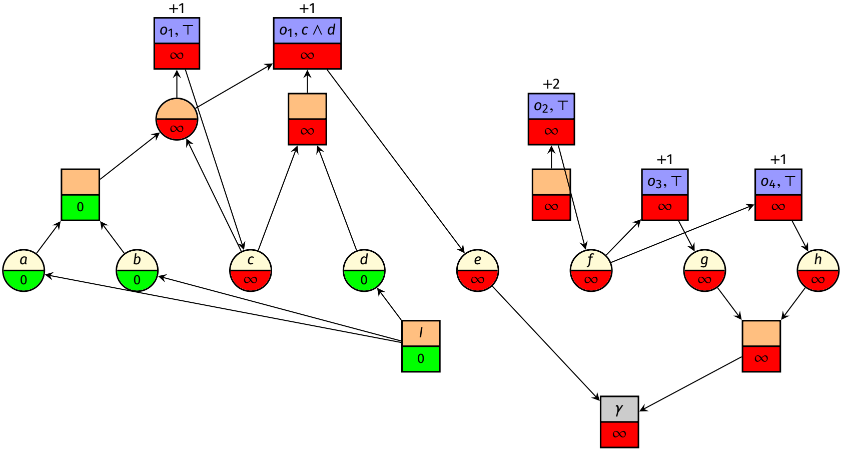
h^{max}: Example



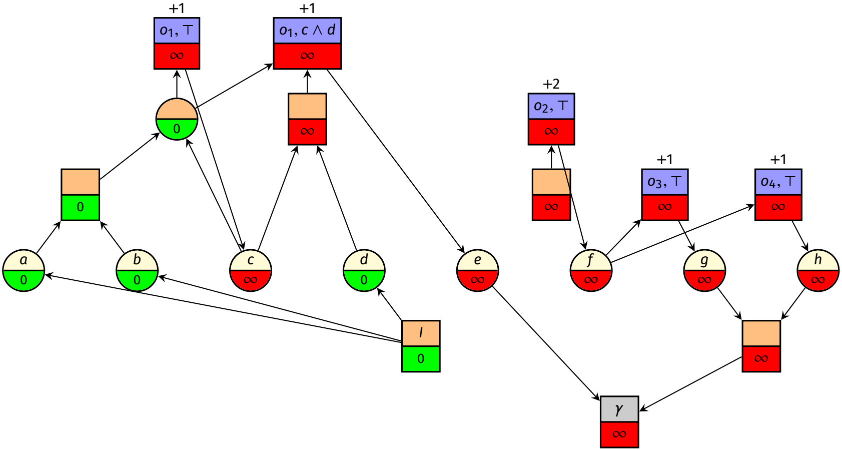
$h^{\max}$: Example



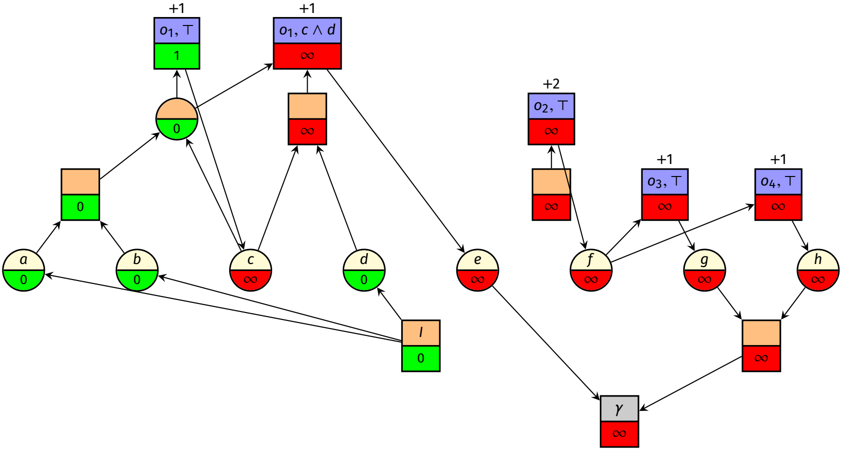
$h^{\max}$: Example



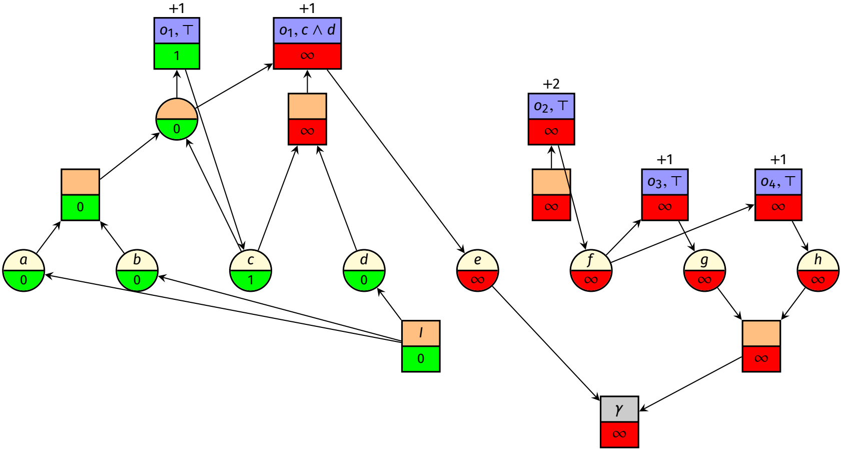
h^{max}: Example



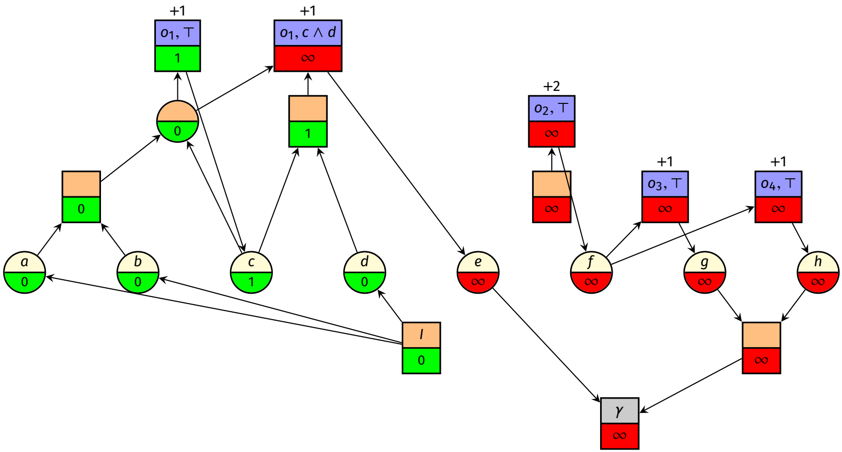
$h^{\max}$: Example



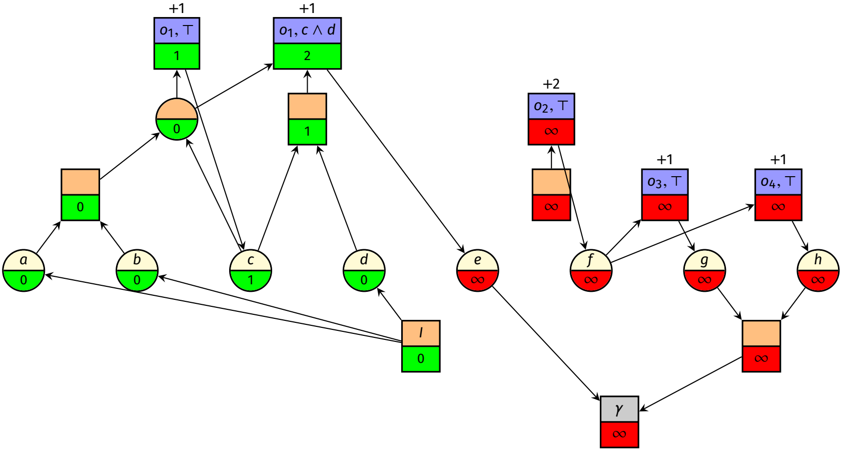
$h^{\max}$: Example



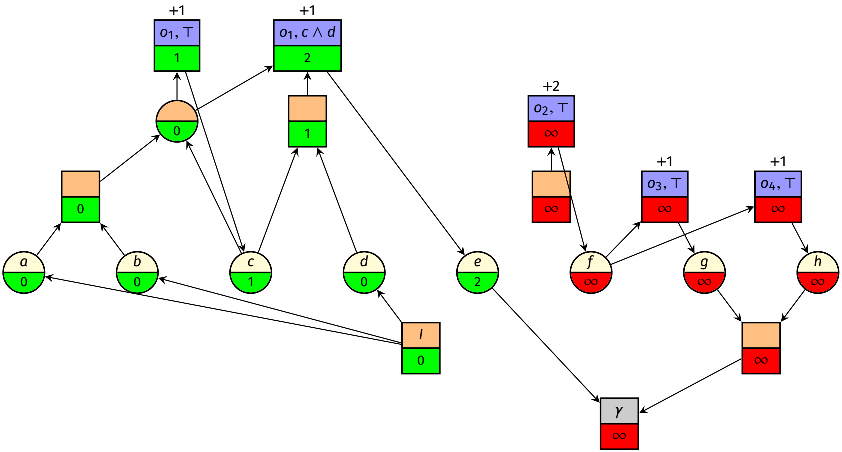
h^{max}: Example



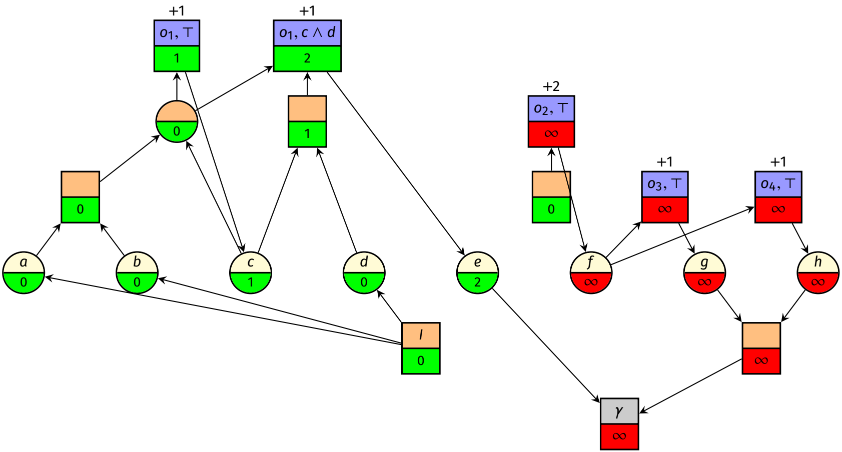
h^{max}: Example



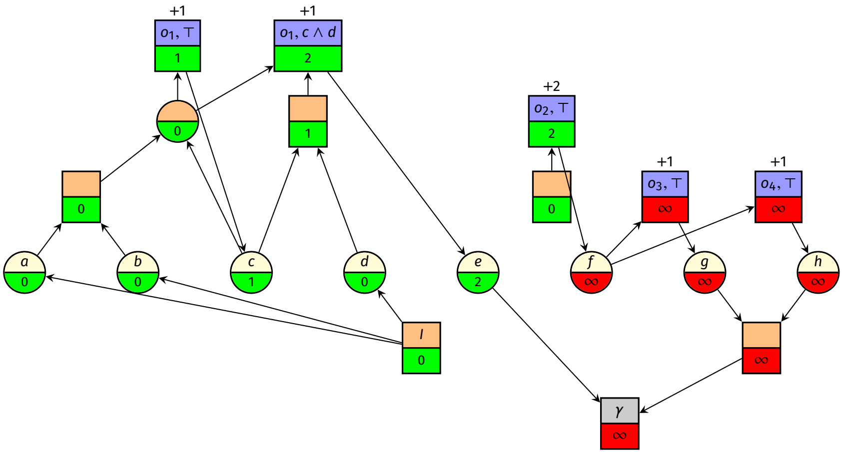
h^{max}: Example



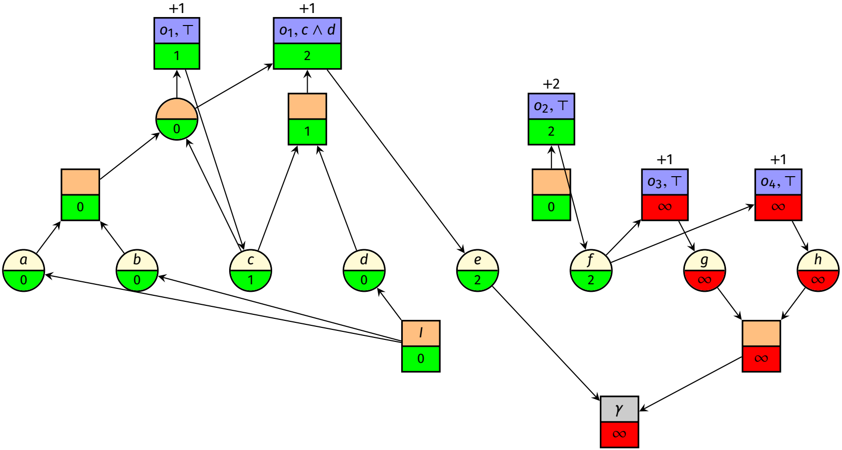
h^{max}: Example



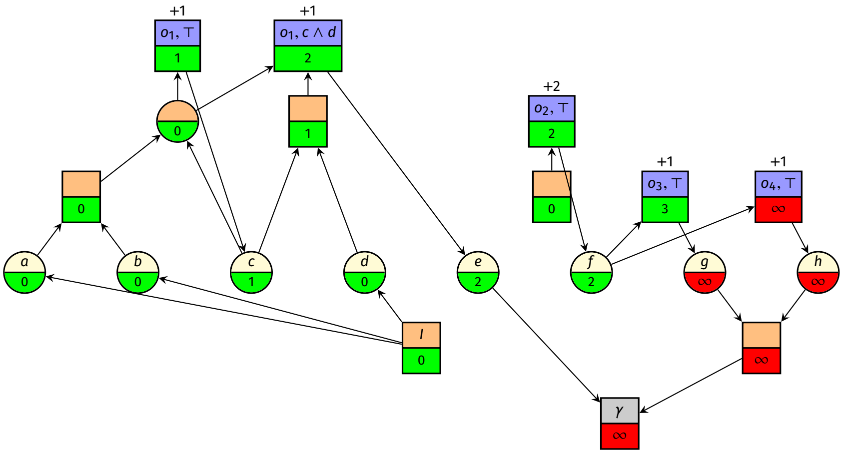
h^{max}: Example



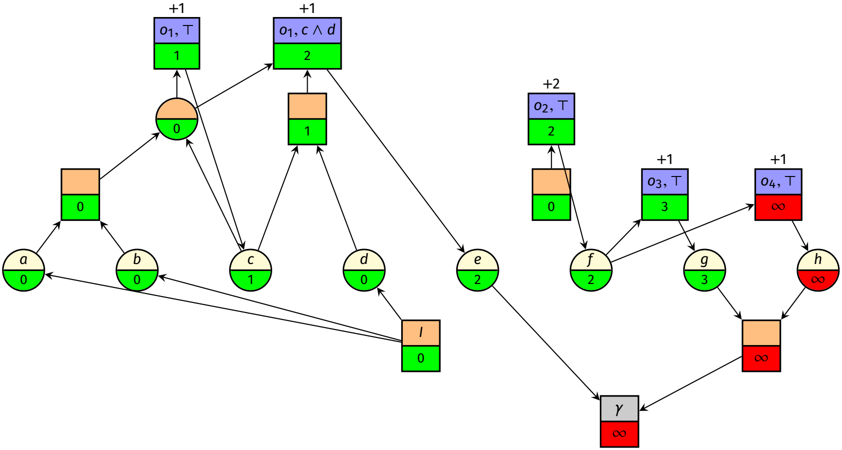
h^{max}: Example



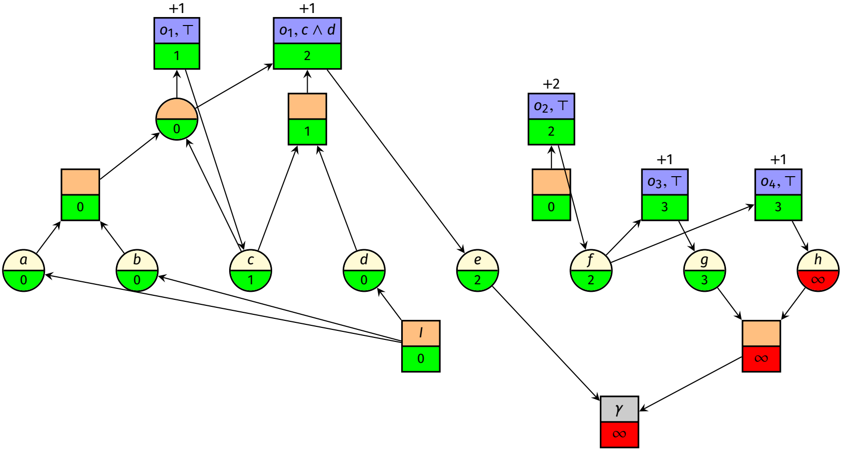
h^{max}: Example



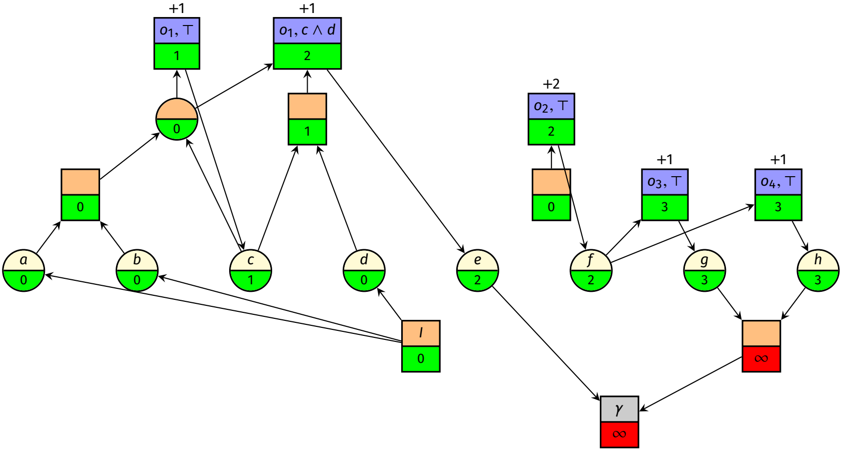
h^{max}: Example



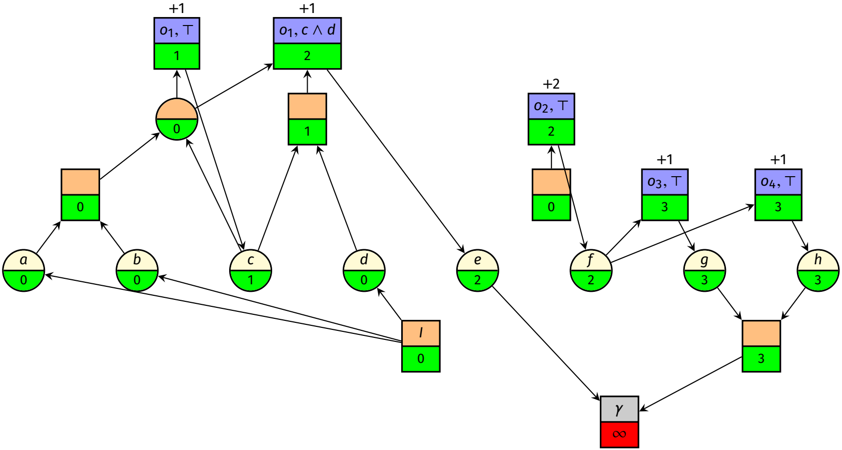
h^{max}: Example



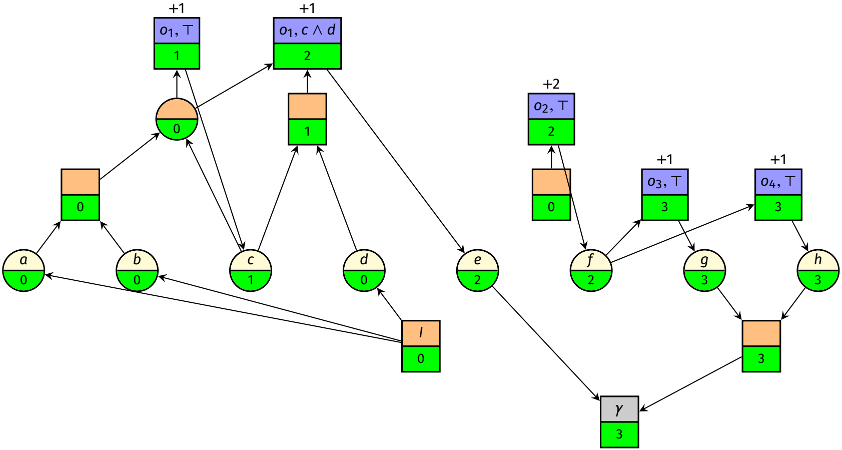
h^{max}: Example



h^{max}: Example



h^{max}: Example



$\rightsquigarrow h^{max}(I) = 3$

h^{add} Algorithm

(Differences to h^{max} algorithm highlighted.)

Computing h^{add} Values

Associate a *cost* attribute with each node.

for all nodes n:

$n.cost := \infty$

while no fixed point is reached:

Choose a node n.

if n is an AND node that is not an effect node:

$n.cost := \sum_{n' \in succ(n)} n'.cost$

if n is an effect node for operator o:

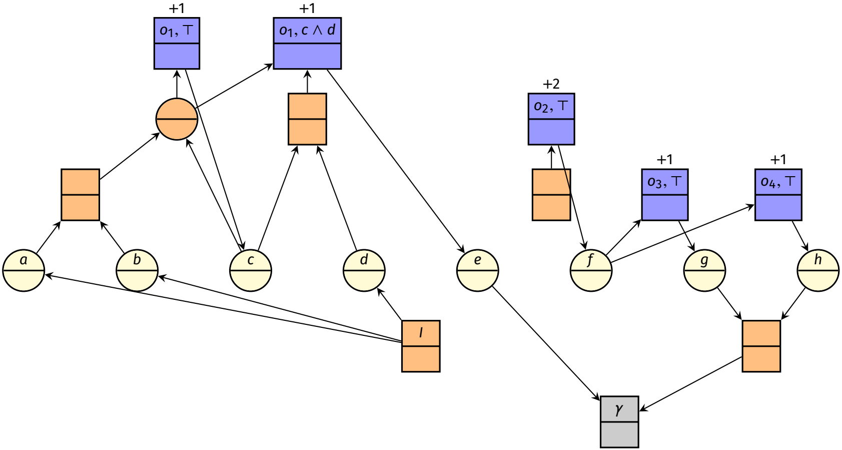
$n.cost := cost(o) + \sum_{n' \in succ(n)} n'.cost$

if n is an OR node:

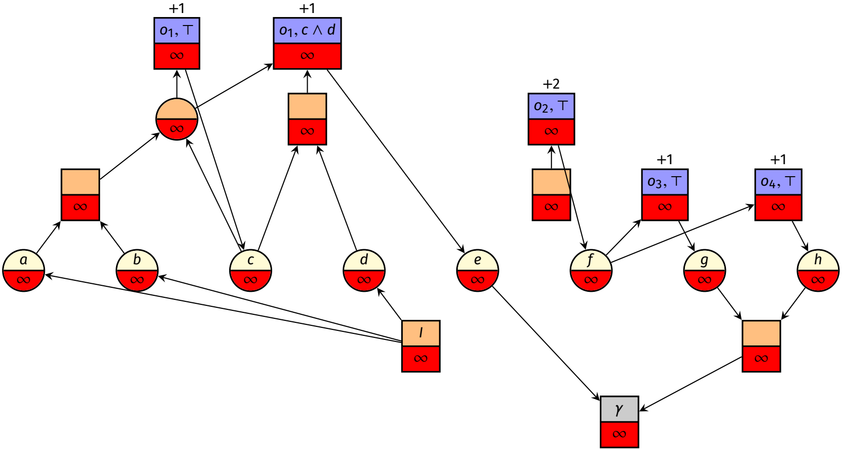
$n.cost := \min_{n' \in succ(n)} n'.cost$

The overall heuristic value is the cost of the goal node, $n_\gamma.cost$.

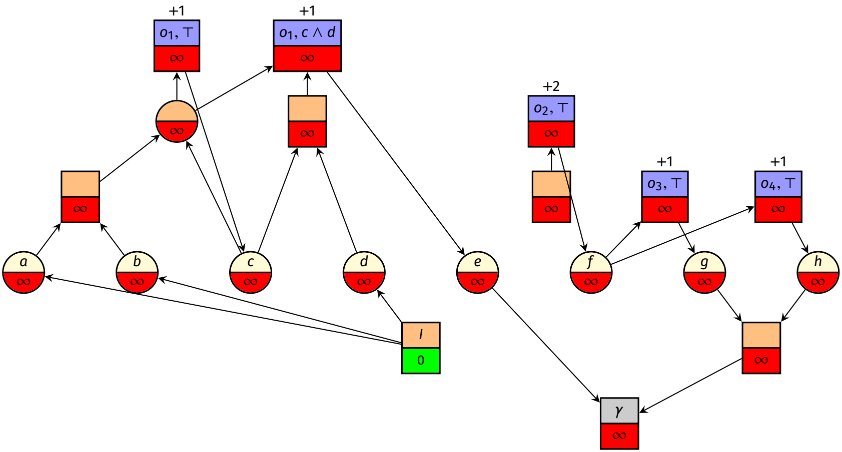
h^{add}: Example



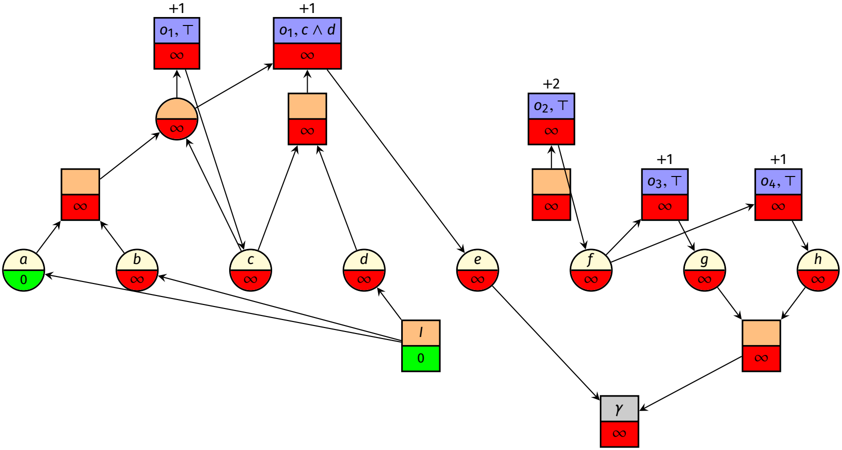
h^{add}: Example



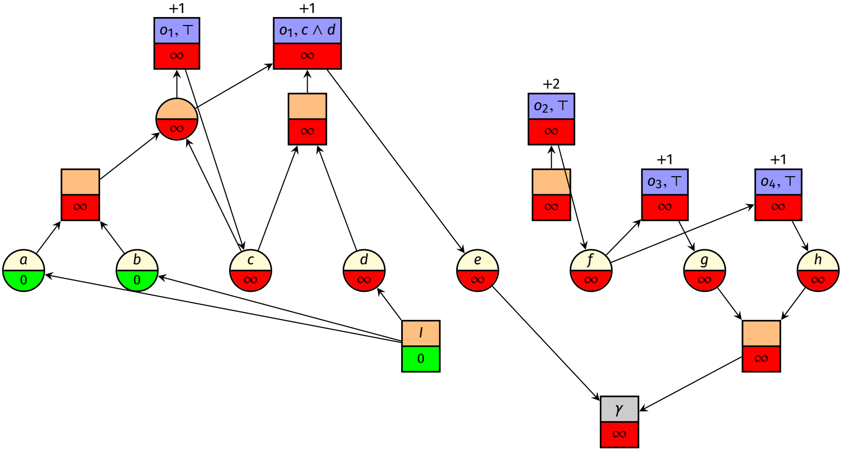
h^{add}: Example



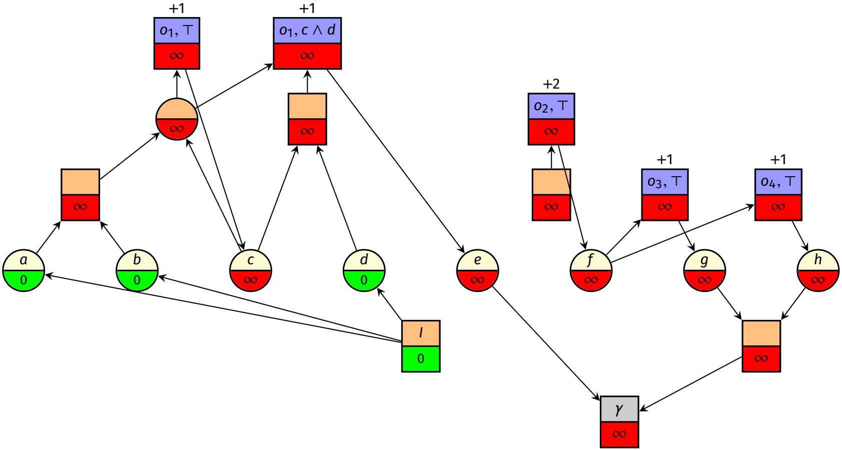
h^{add}: Example



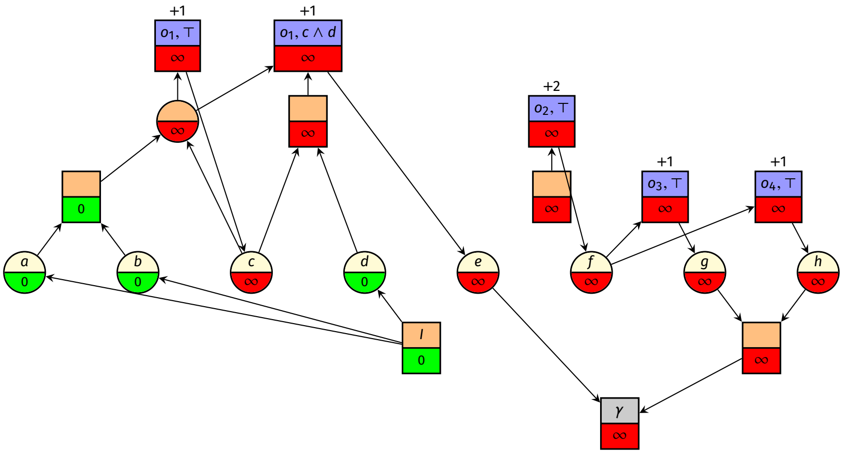
h^{add}: Example



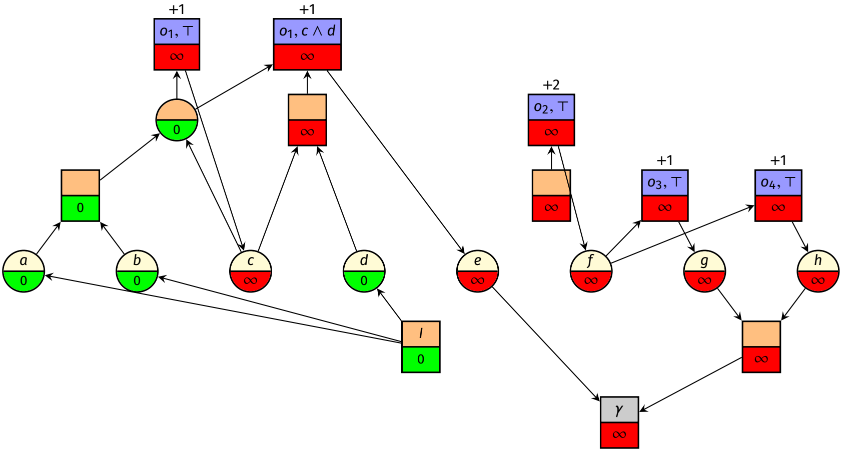
h^{add}: Example



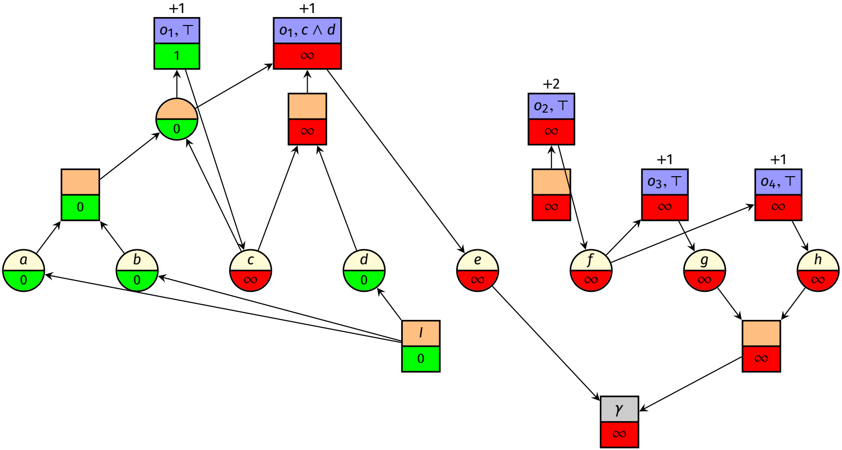
h^{add}: Example



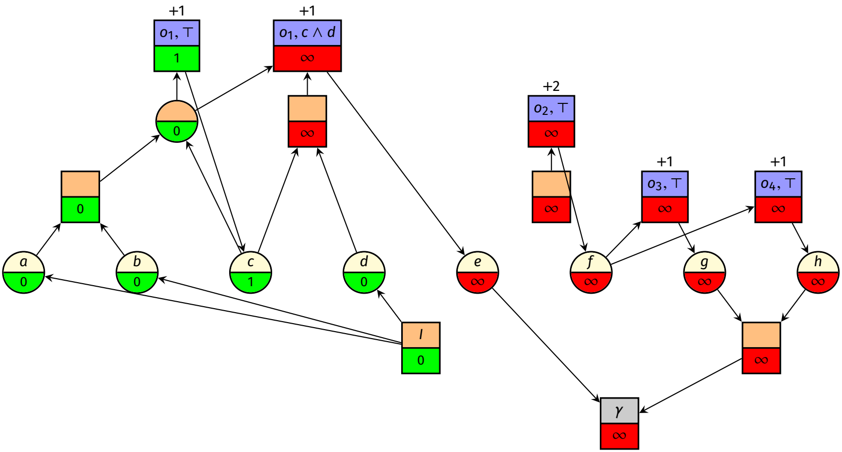
h^{add}: Example



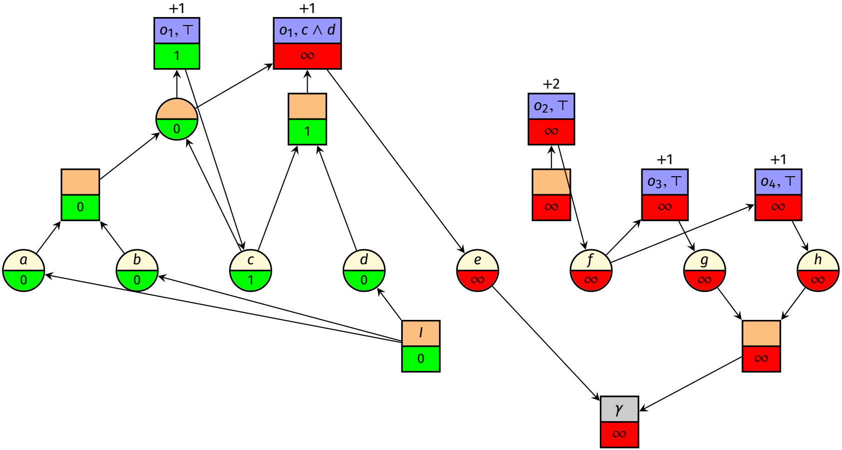
h^{add}: Example



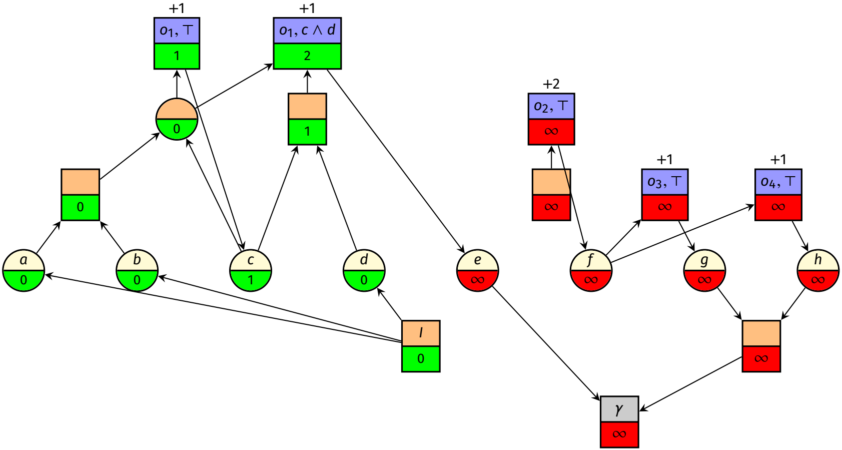
h^{add}: Example



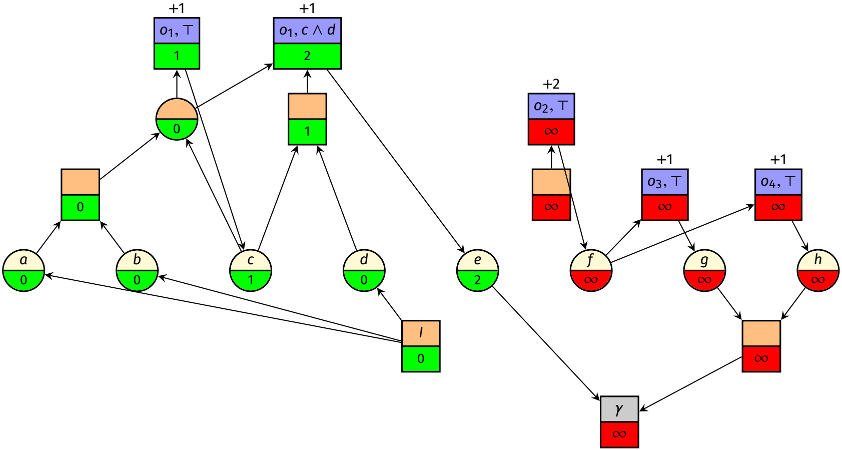
h^{add}: Example



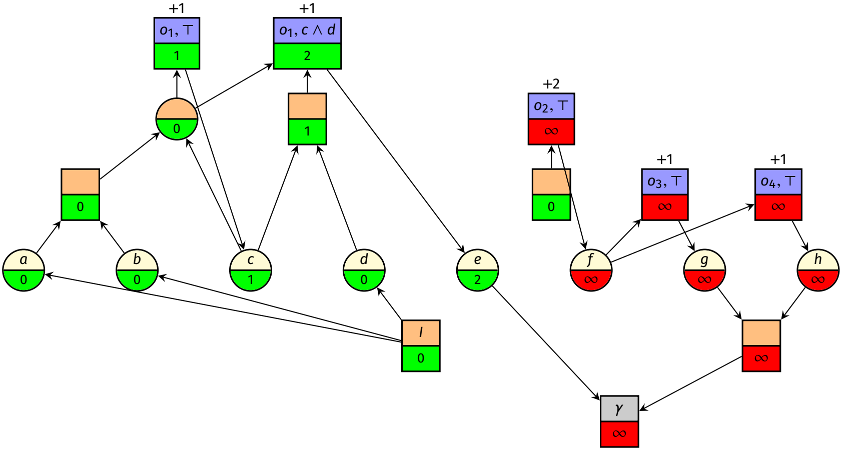
h^{add}: Example



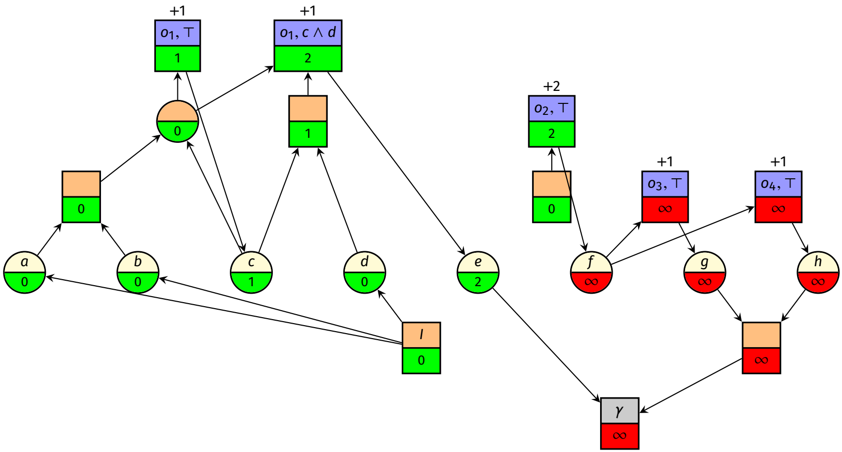
h^{add}: Example



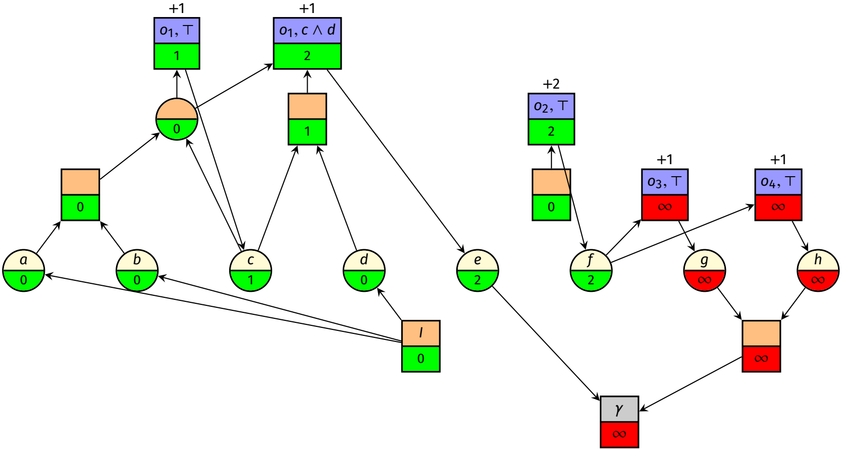
h^{add}: Example



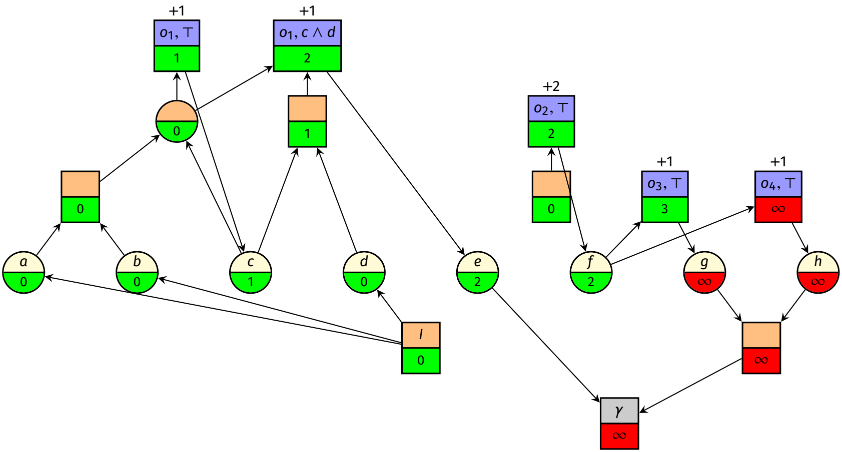
h^{add}: Example



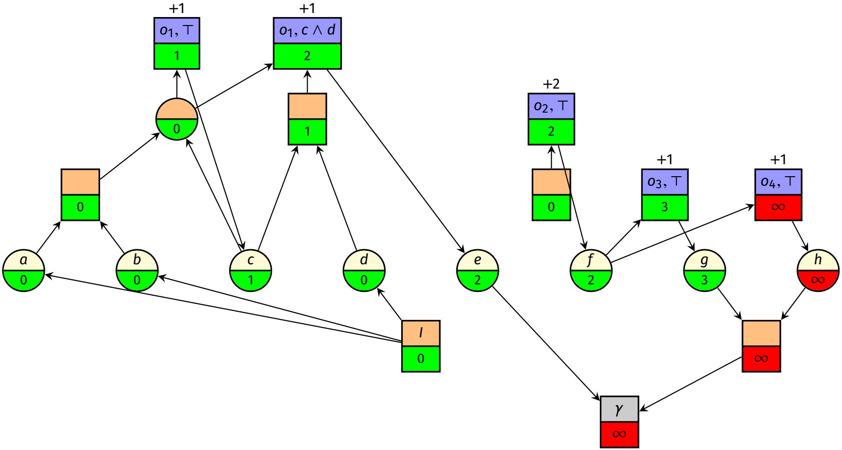
h^{add}: Example



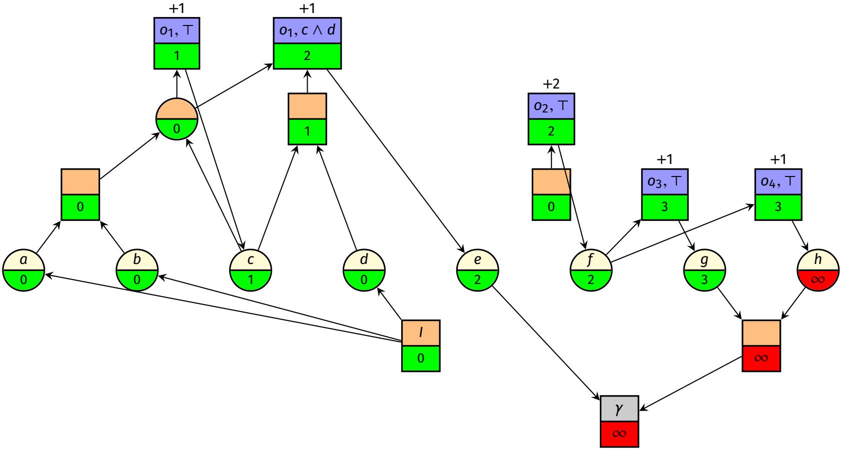
h^{add}: Example



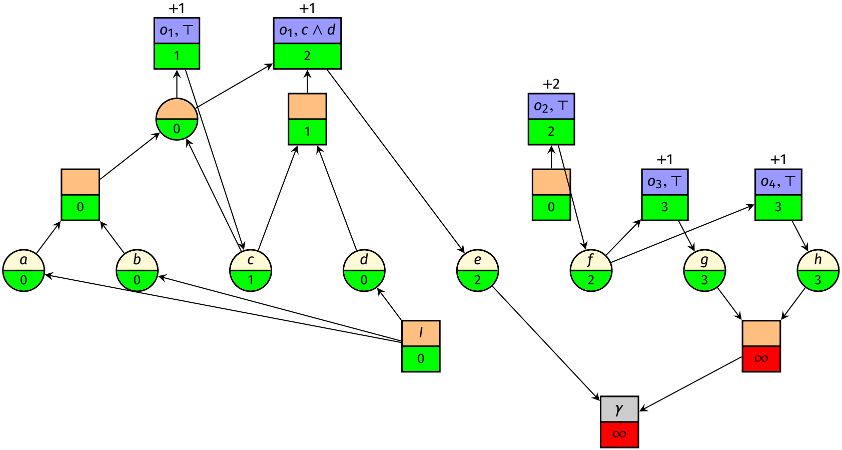
h^{add}: Example



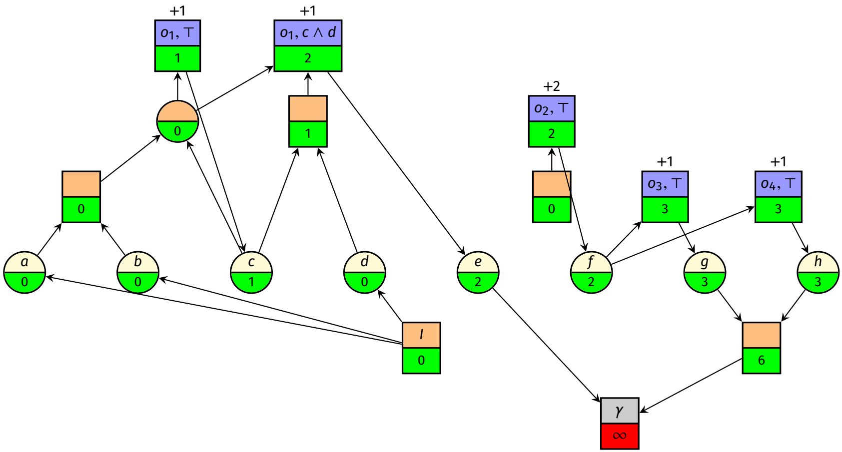
h^{add}: Example



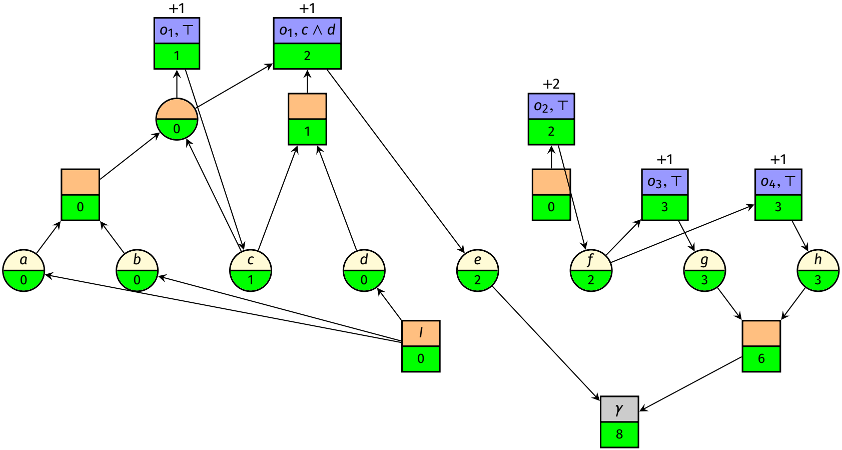
h^{add}: Example



h^{add}: Example



h^{add}: Example



$\rightsquigarrow h^{add}(I) = 8$

h^{max} and h^{add}: Definition

We can now define our first non-trivial efficient planning heuristics:

h^{max} and h^{add} Heuristics

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a propositional planning task in positive normal form.

The h^{max} heuristic value of a state s, written $h^{max}(s)$, is obtained by constructing the RTG for $\Pi_s^+ = \langle V, s, O^+, \gamma \rangle$ and then computing $n_\gamma.cost$ using the h^{max} value algorithm for RTGs.

The h^{add} heuristic value of a state s, written $h^{add}(s)$, is computed in the same way using the h^{add} value algorithm for RTGs.

Notation: we will use the same notation $h^{max}(n)$ and $h^{add}(n)$ for the h^{max}/h^{add} values of RTG nodes

Properties of h^{max} and h^{add}

Understanding h^{max} and h^{add}

We want to understand h^{max} and h^{add} better:

- Are they well-defined?
- How can they be efficiently computed?
- Are they safe?
- Are they admissible?
- How do they compare to the optimal solution cost for a delete-relaxed task (h^+)?

Understanding h^{max} and h^{add}

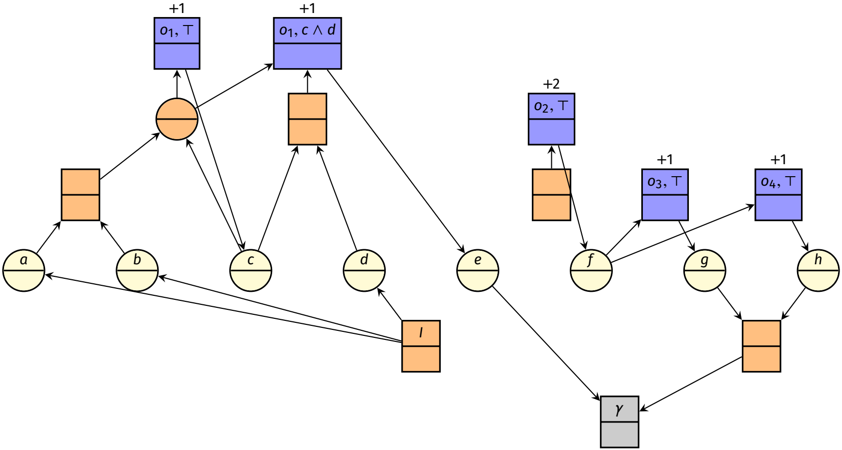
We want to understand h^{max} and h^{add} better:

- Are they well-defined? Yes.
- How can they be efficiently computed?
- Are they safe?
- Are they admissible?
- How do they compare to the optimal solution cost for a delete-relaxed task (h^+)?

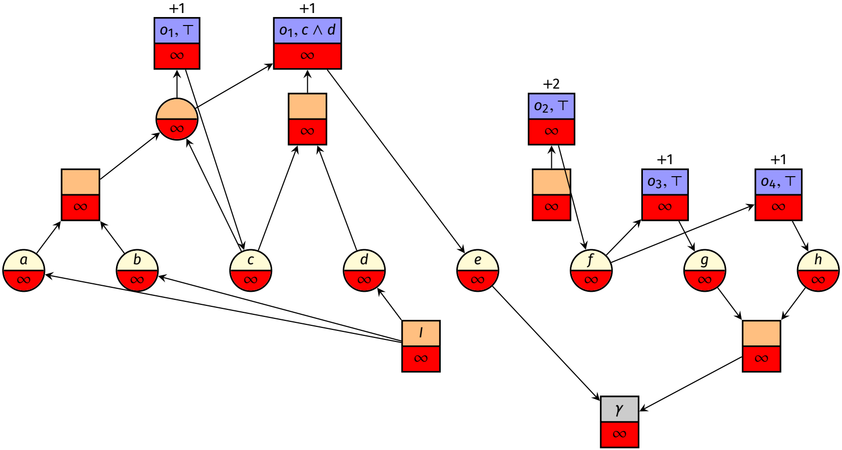
Efficient Computation of h^{max} and h^{add}

- If nodes are poorly chosen, the h^{max}/h^{add} algorithm can update the same node many times until it reaches its final value.
- However, there is a simple strategy that prevents this: in every iteration, pick a node with minimum new value among all nodes that can be updated to a new value.
- With this strategy, no node is updated more than once.
- Using a suitable priority queue data structure, this allows computing the h^{max}/h^{add} values of an RTG with nodes N and arcs A in time $O(|N| \log |N| + |A|)$.

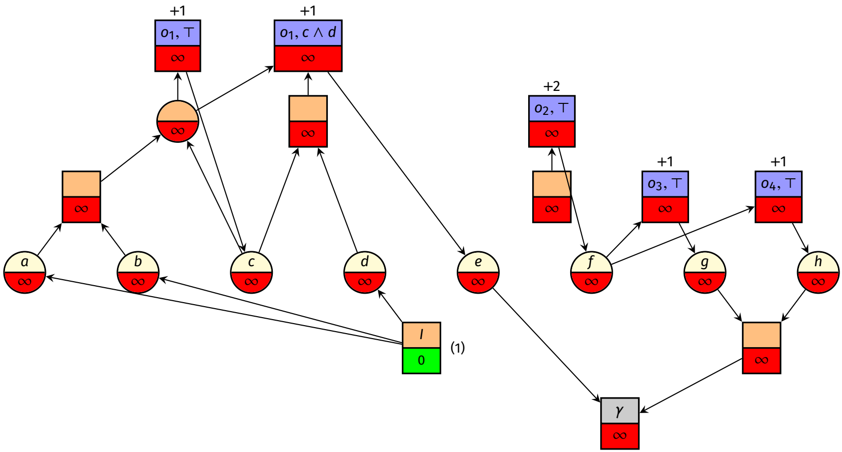
h^{max}: Example of Efficient Computation



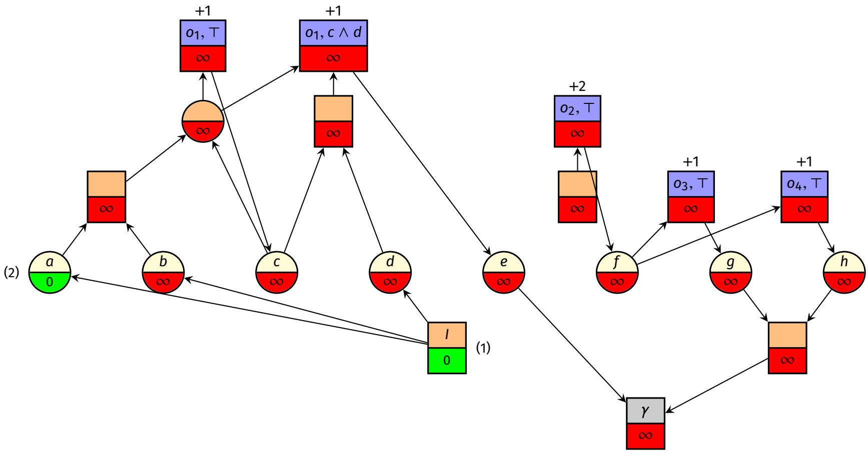
h^{max}: Example of Efficient Computation



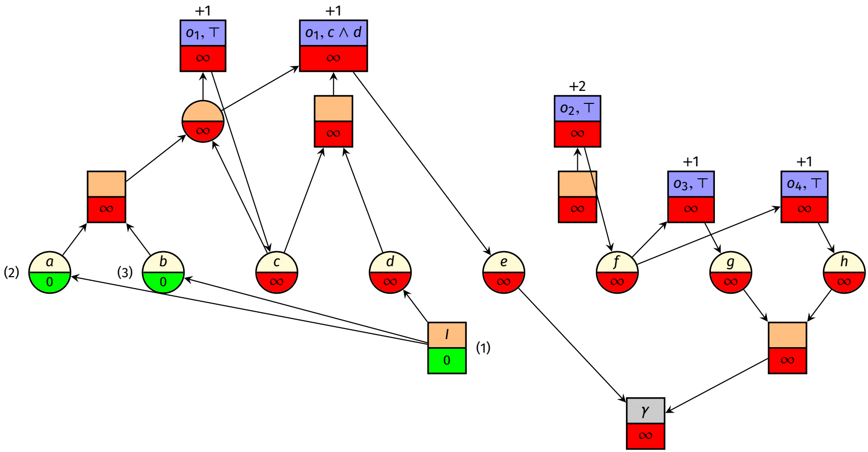
h^{max}: Example of Efficient Computation



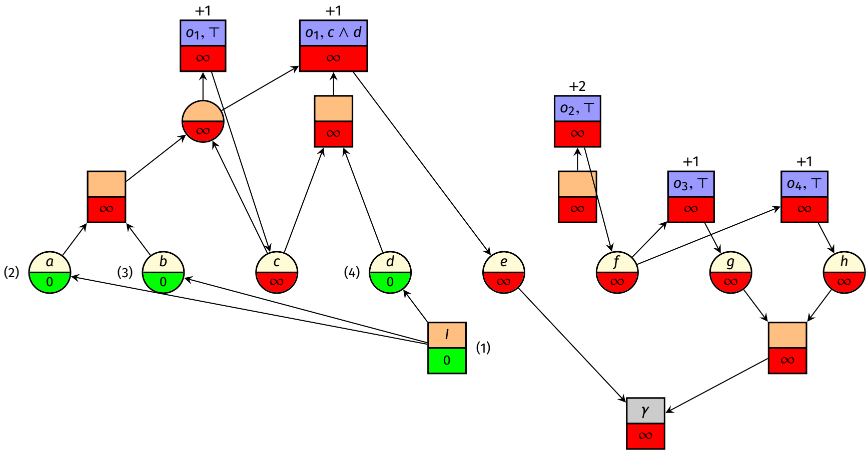
h^{max}: Example of Efficient Computation



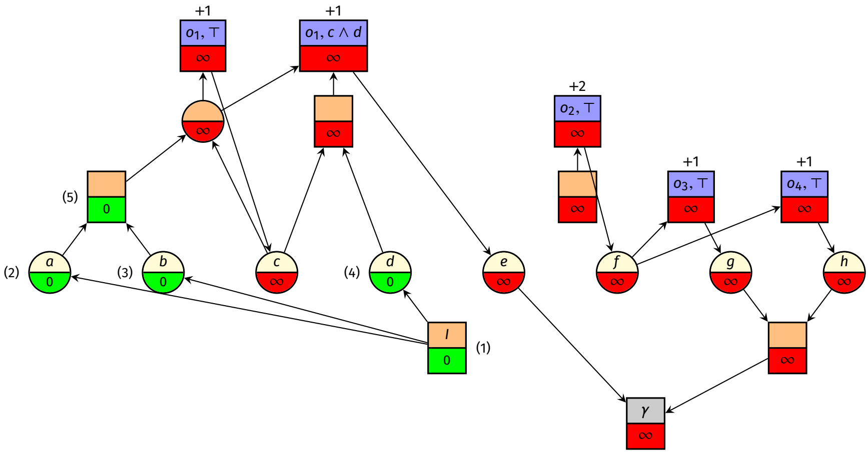
h^{max}: Example of Efficient Computation



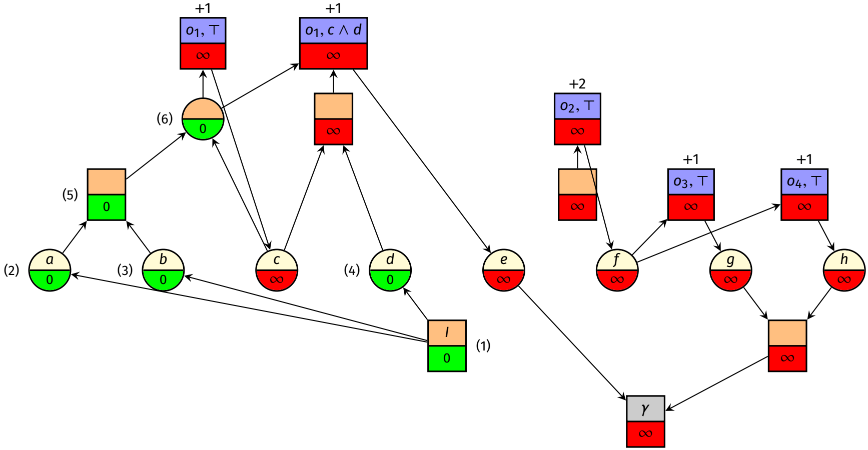
h^{max}: Example of Efficient Computation



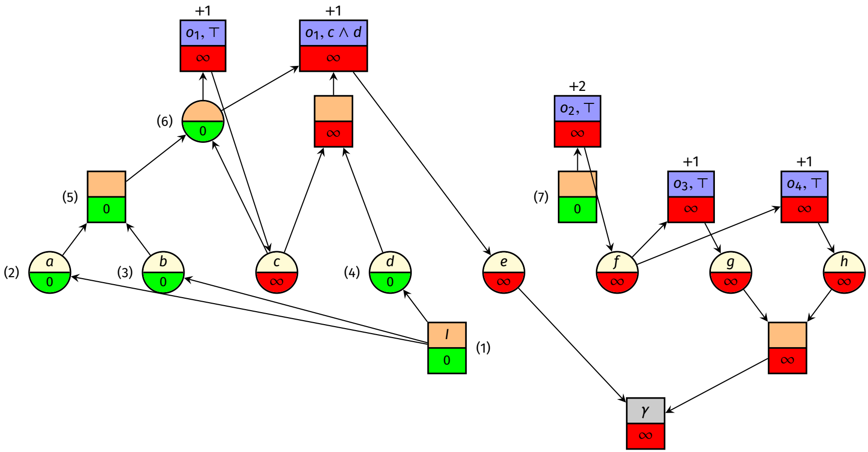
h^{max}: Example of Efficient Computation



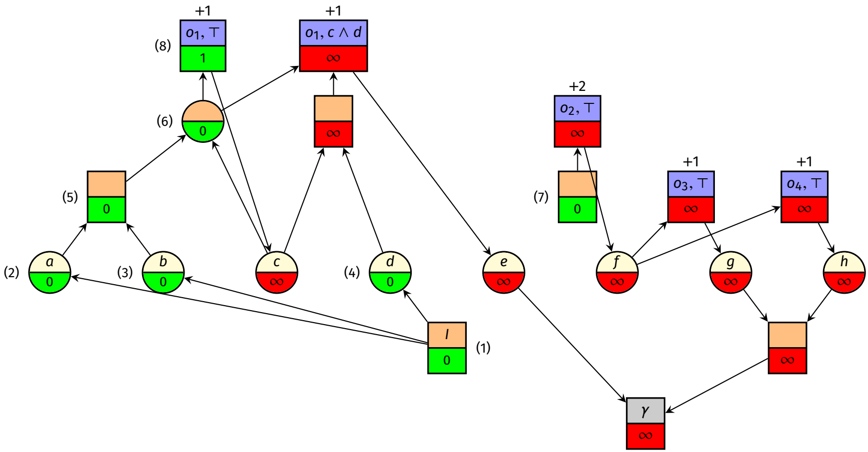
h^{max}: Example of Efficient Computation



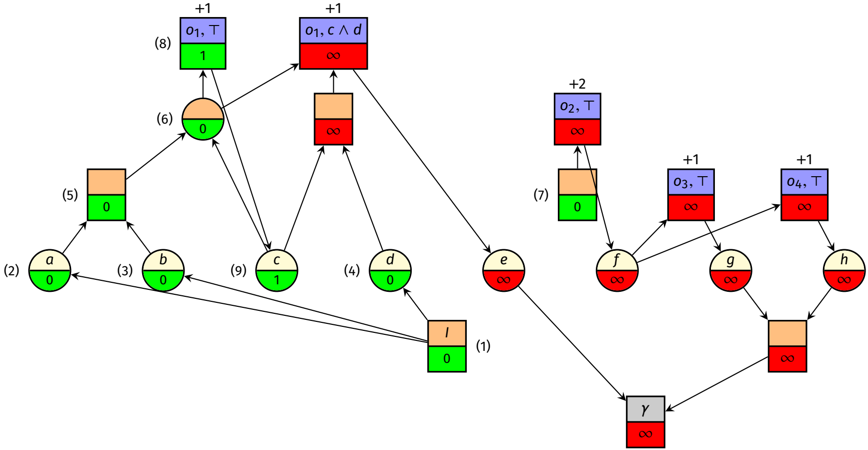
h^{max}: Example of Efficient Computation



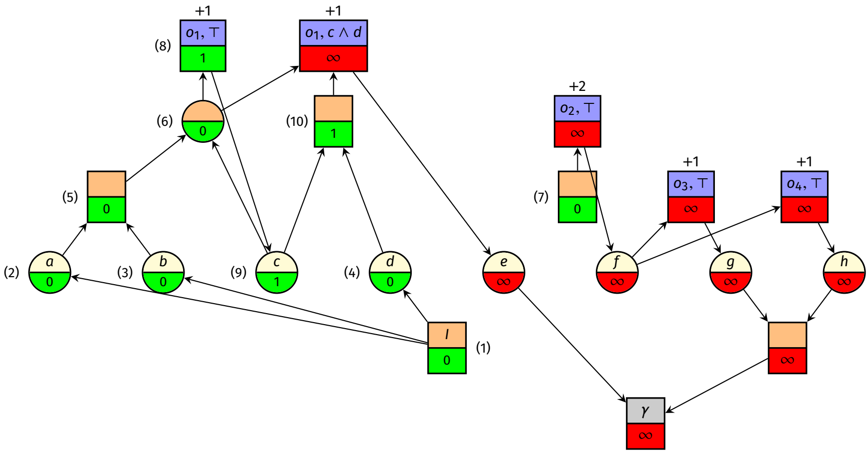
$h^{\max}$: Example of Efficient Computation



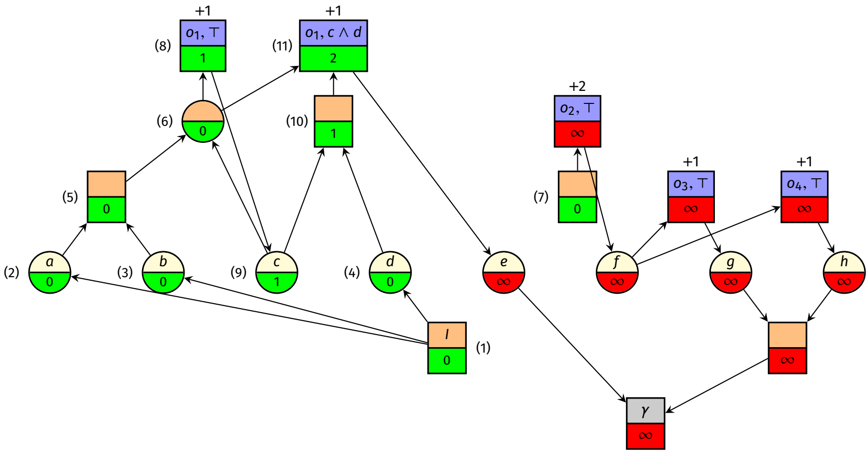
h^{max}: Example of Efficient Computation



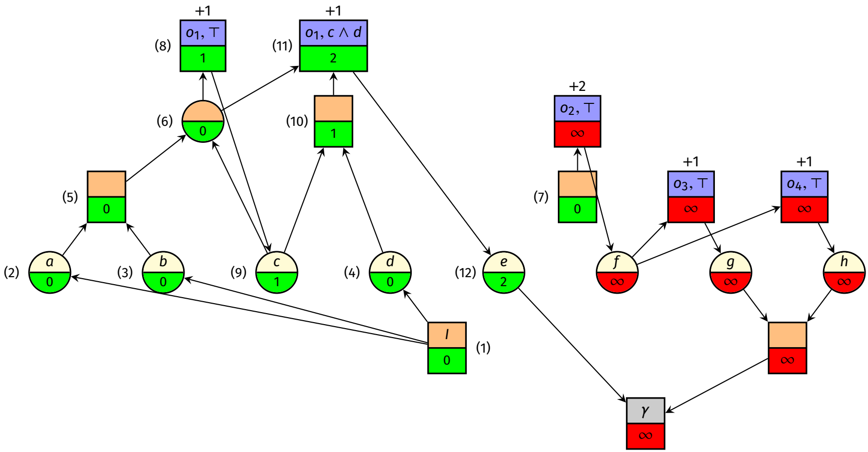
h^{max}: Example of Efficient Computation



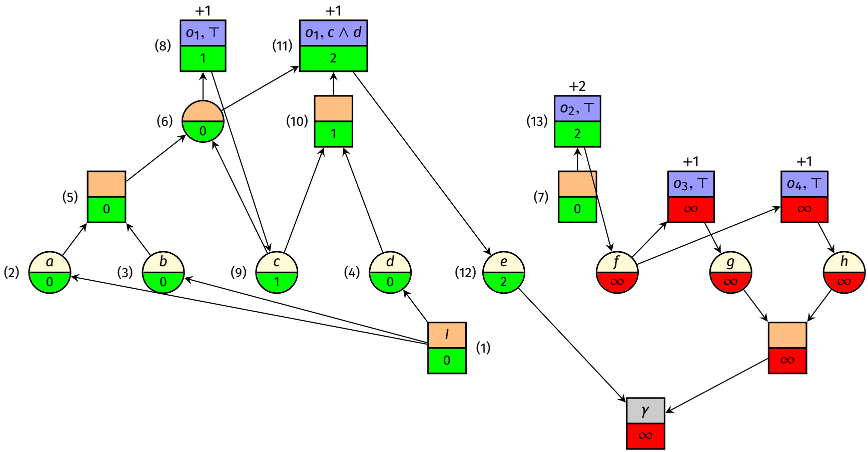
h^{max}: Example of Efficient Computation



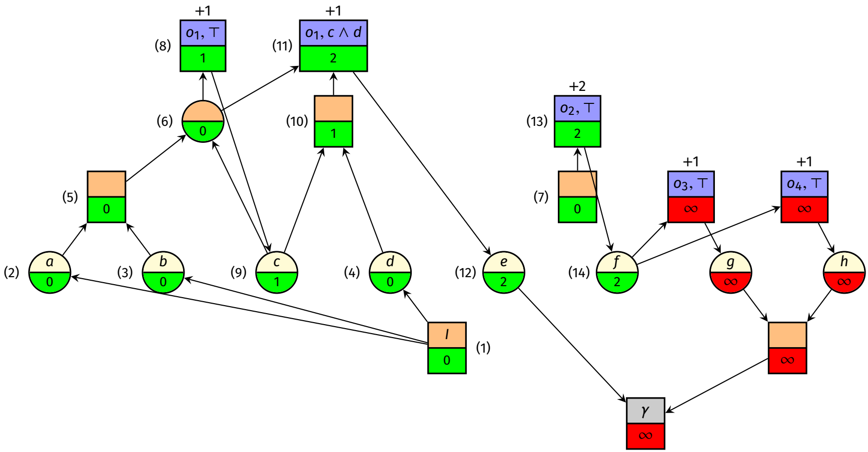
h^{max}: Example of Efficient Computation



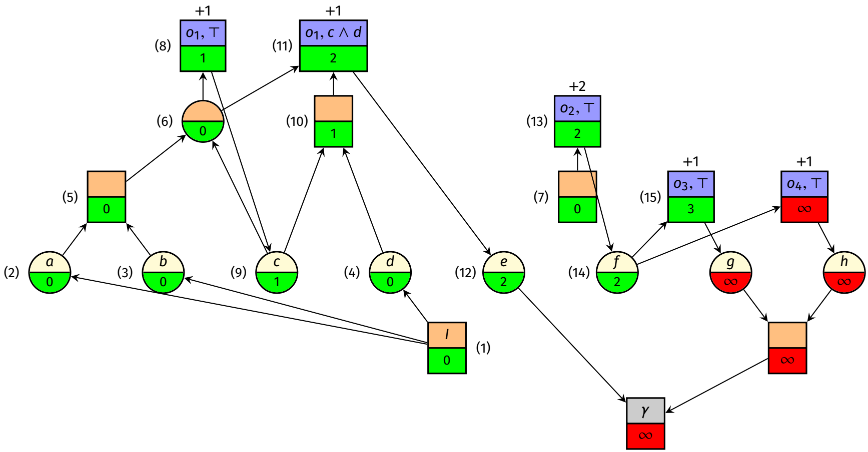
$h^{\max}$: Example of Efficient Computation



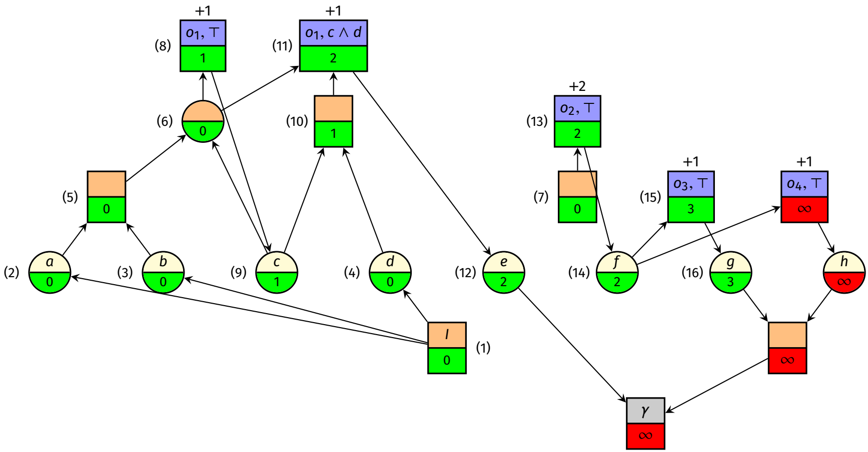
h^{max}: Example of Efficient Computation



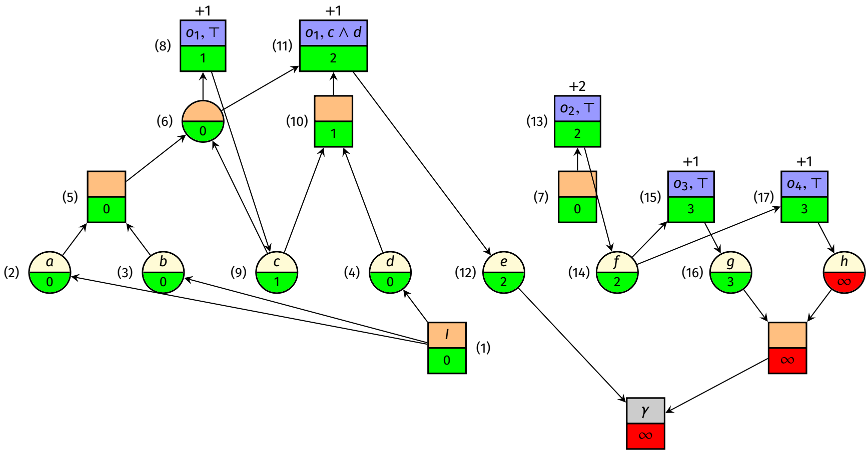
h^{max}: Example of Efficient Computation



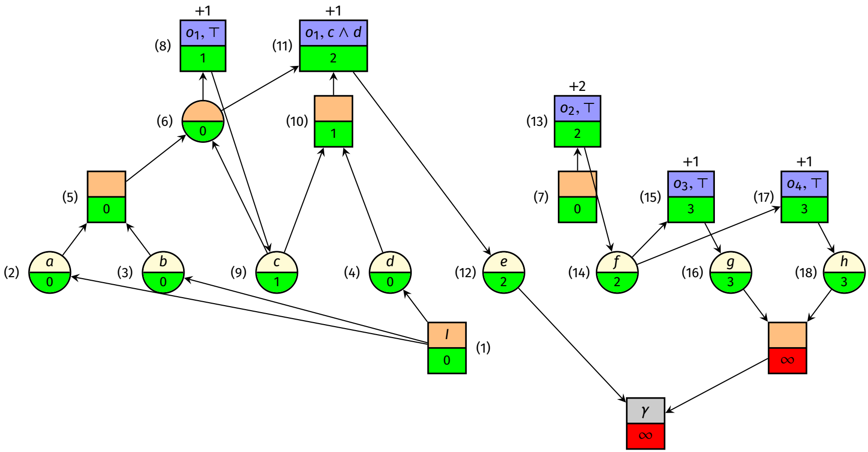
h^{max}: Example of Efficient Computation



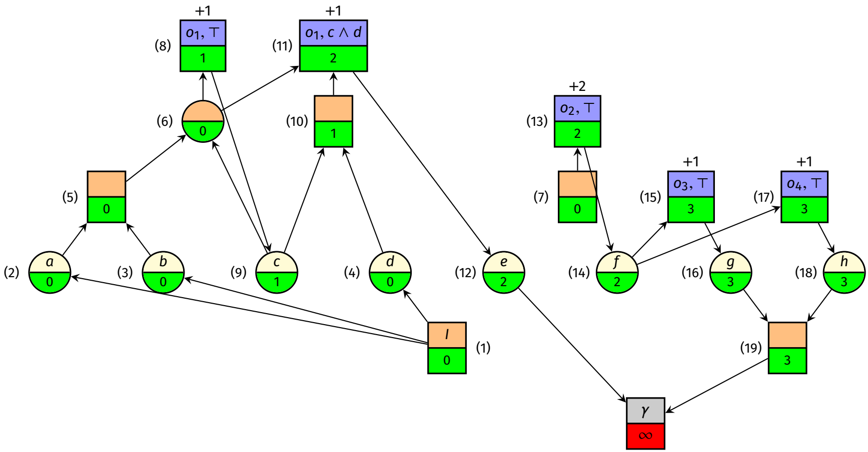
h^{max}: Example of Efficient Computation



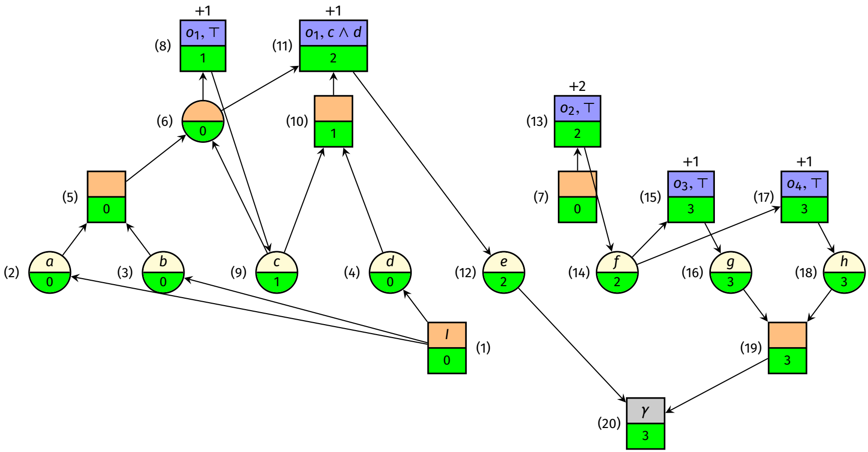
h^{max}: Example of Efficient Computation



h^{max}: Example of Efficient Computation



$h^{\max}$: Example of Efficient Computation



$\rightsquigarrow h^{\max}(I) = 3$

Efficient Computation of h^{max} and h^{add}: Remarks

- In the following chapters, we will always assume that we are using this efficient version of the h^{max} and h^{add} algorithm.
- In particular, we will assume that all reachable nodes of the relaxed task graph are processed exactly once (and all unreachable nodes not at all), so that it makes sense to speak of certain nodes being processed after others etc.

Heuristic Quality of $h^{\max}$ and h^{add}

This leaves us with the questions about the heuristic quality of $h^{\max}$ and h^{add}:

- Are they safe?
- Are they admissible?
- How do they compare to the optimal solution cost for a delete-relaxed task?

It is easy to see that $h^{\max}$ and h^{add} are safe: they assign ∞ iff a node is unreachable in the delete relaxation.

In our running example, it seems that $h^{\max}$ is prone to underestimation and h^{add} is prone to overestimation.

We will study this further in the next chapter.

Summary

Summary

- h^{max} and h^{add} values estimate how expensive it is to reach a state variable, operator effect or formula (e.g., the goal).
- They are computed by propagating cost information in relaxed task graphs:
  - At OR nodes, choose the cheapest alternative.
  - At AND nodes, maximize or sum the predecessor costs.
  - At effect nodes, also add the operator cost.
- h^{max} and h^{add} values can serve as heuristics.
- They are well-defined and can be computed efficiently by computing them in order of increasing cost along the RTG.