# Planning for Artificial Intelligence

**Lukáš Chrpa**

Relaxation Heuristics

# Mutexes, Search, Heuristics

(revision)

# Mutual Exclusivity (Mutex)

- We say that atoms (or facts) p and q are **mutually exclusive** (or **mutex**) in a given planning task iff for each reachable state s, $\{p,q\} \not\subseteq s$
- We say that a set of atoms $\{p_1,p_2,\ldots,p_n\}$ forms a **mutex group** in a given planning task iff for each reachable state s, $|\{p_1,p_2,\ldots,p_n\} \cap s| \leq 1$
- We say that a set of atoms $\{p_1,p_2,\ldots,p_n\}$ forms a **facts alternating mutex (FAM) group** in a given planning task iff for each reachable state s, $|\{p_1,p_2,\ldots,p_n\} \cap s| = 1$
- FAM group is a special case of Mutex group
- Atoms in a mutex group are pairwise mutex

# Mutex - examples

- Logistics
  - {at-truck-A, at-truck-B, at-truck-C} is a mutex group
  - at-package-A, in-truck-package are mutex
- BlocksWorld
  - {on-A-B,ontable-A,holding-A} is a mutex group
  - on-A-B and clear-B are mutex
- Sokoban
  - {at-box1-5-5,at-person-5-5-,free-5-5} is a mutex group
  - at-box1-5-5 and at-box1-6-6 are mutex

# Informed Search

- Systematic (one-directional)
  - Greedy Best First Search (GBFS)
  - A*
  - Weighted A*
- Systematic bidirectional
- Local
  - (Enforced) Hill Climbing

# Heuristic Function

- Let $S$ be a set of states for a given planning task $\Pi$. A **heuristic function** (or **heuristics**) for $\Pi$ is a function $h{:}S \rightarrow N_0 \cup \{\infty\}$
- The value $h(s)$ **estimates** distance from $s$ to the nearest goal state
- $h(s)$ is called **heuristic estimate** or **heuristic value** for $s$
- A **perfect** (or optimal) **heuristics**, denoted as $h^*$, maps each state to the length (or cost) of the optimal plan to the nearest goal state
  - If $h^*(s)=\infty$ then no goal state is reachable from $s$

# Properties of Heuristic Function

- Heuristic function $h$ for $\Pi$ (over $S$) is
  - **safe** if for each $s \in S$ s.t. $h(s)=\infty$ it holds that $h^*(s)=\infty$
  - **goal aware** if $h(s_G)=0$ for each goal state $s_G$
  - **admissible** if for each $s \in S$ it holds that $h(s) \leq h^*(s)$
  - **consistent** if goal aware and for each $s,s' \in S$ s.t. $s'$ is a successor of $s$ it holds that $h(s) \leq h(s')+cost(s,s')$

# Towards Good Heuristics

# Ideal Properties of Heuristics

- **Easy to compute** (at most in linear time)
- **Easy to implement**
- **Very informative** (close to the perfect heuristic)

- These properties often go against each other

# Goal Count Heuristic

- The **Goal Count heuristic** represents how many goal atoms have yet to be achieved
- $h_G(s)=|G \setminus s|$
- Easy to compute ?

# Goal Count Heuristic

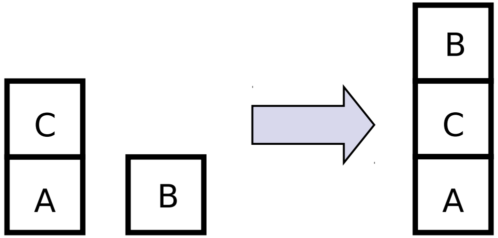
- The **Goal Count heuristic** represents how many goal atoms have yet to be achieved
- $h_G(s)=|G \setminus s|$
- Easy to compute ?
  - Yes
- Easy to implement ?

# Goal Count Heuristic

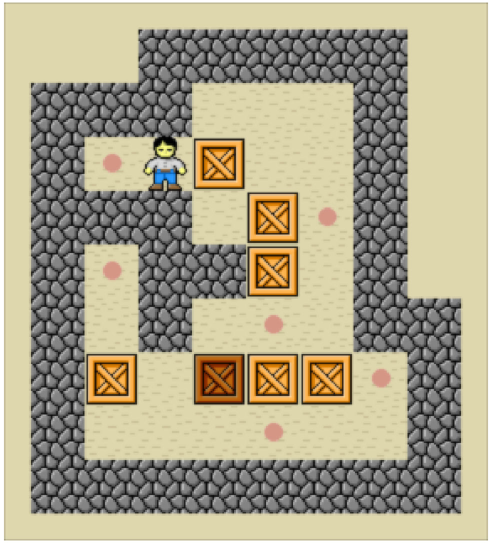
- The **Goal Count heuristic** represents how many goal atoms have yet to be achieved
- $h_G(s)=|G \setminus s|$
- Easy to compute ?
  - Yes
- Easy to implement ?
  - Yes
- Informative ?

# Goal Count Heuristic - Issues

- Some goals are achieved too early
  - Sussman anomaly (in BW)
- If the goal has only one atom
- It might take many steps to achieve one goal atom
  - e.g. Sokoban
- Not admissible
  - one action can achieve more goal atoms




The goal is to to build the A-B-C tower

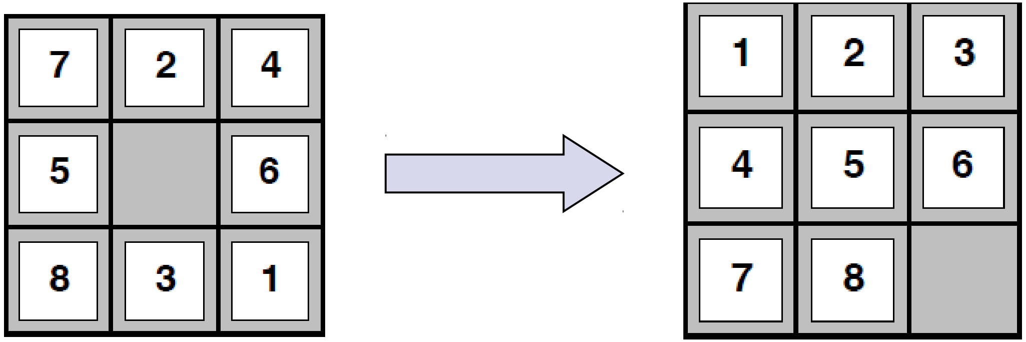


# How to effectively compute reasonably informative heuristics ?

- **Relax** some problem constraints
- **Abstract** the problem
- Leverage some **structural information**
  - **Landmarks**
  - **Potentials**
- ….

# Relaxation

# 8-puzzle example




- A tile can move from square A to B if A is adjacent to B and B is free $\rightarrow$ $h^*$
- A tile can move from square A to B if A is adjacent to B $\rightarrow$ $h^{MD}$ (Manhattan distance)
- A tile can move from square A to B $\rightarrow$ $h^{MT}$ (Misplaced Tiles)

- $h^* \geq h^{MD} \geq h^{MT}$ (why?)

# Relaxation

- **Removing one or more constraints** from the problem
- **Solution** of the **original** problem is a **solution** of the **relaxed** problem
- If the **relaxed** problem is **unsolvable**, then the **original** problem is **unsolvable** too
- Solving the relaxed problem is at most as hard as solving the original problem

# Relaxation in planning

- How to relax planning tasks ?
  - remove delete effects !
  - in SAS, don’t remove variable assignment when its value changes (cumulate the values)
- We sometimes explicitly refer to such a relaxation as **delete-relaxation**

# Relaxed Planning Tasks

- The (delete-)**relaxation** $a^+$ of an action a=(pre(a),del(a),add(a)) is $a^+$=(pre(a),add(a))
- The **result** of application of $a^+$ in a state s (if possible) is s'=s∪add(a)
- Let Π=(P,A,I,G) be a planning task. The **relaxed planning task** $\Pi^+$ for Π is $\Pi^+$=(P,{$a^+$ | a∈A},I,G)
- If $\pi^+$ is a plan for $\Pi^+$, then $\pi^+$ is a **relaxed plan** for Π

- A **perfect** (or optimal) **relaxed heuristics**, denoted as $h^+$, maps each state to the length (or cost) of the optimal relaxed plan to the nearest goal state

# $h^+$

- $h^+$ is **safe, goal aware, admissible** and **consistent**
- Finding optimal (delete-)relaxed plans is **NP-hard**
  - Not very practical to use $h^+$
- Any other idea ?

# Greedy Algorithm for Relaxed Planning Tasks

$s := I$

$\pi^+ := \langle\rangle$

**while** $G \not\subseteq s$ **do**

    select any $a^+ \in A^+$ s.t. $a^+$ is applicable in $s$ and $add(a) \not\subseteq s$

    **if** no such $a^+$ exists **then return** no solution

    $s := s \cup add(a)$

    $\pi^+ := \pi^+ . a^+$

**return** $\pi^+$

# Properties of the Algorithm

- **sound**
  - returned plan is a relaxed plan for the planning task
  - if "unsolvable" is returned, then no action can add an atom to the state and hence some goal atoms cannot be achieved
- **complete**
  - the algorithm always terminates
    - each action can be applied at most once
    - at least one atom is added in each iteration
- **linear** time complexity

# Heuristic from the Greedy Algorithm

- The length or the cost of the relaxed plan (from the state s) is the heuristic value for s
- Such a heuristic is
  - **safe**
  - **goal aware**
- Often such relaxed plans are very suboptimal and such a heuristic is thus not very informative

# Two possibilities how to calculate relaxed heuristics

- Do not generate relaxed plans but **estimate difficulty** of a relaxed planning task
  - $\mathbf{h_{max}}$
  - $\mathbf{h_{add}}$
- Generate “**reasonable” relaxed plans**
  - $\mathbf{h_{FF}}$

# Optimistic and Pessimistic Assumptions of Task Difficulty

- The idea is to **estimate cost** of achieving an **atom** or apply an **action**
- For each **atom** we look for the **cheapest action** to achieve it
- For each **action** we consider (either)
  - **sum** of the costs of the **atoms** in its precondition ($h_{add}$)
  - **maximum** of the costs of the **atoms** in its precondition ($h_{max}$)
- It can be observed that
  - $\mathbf{h_{max}}$ provides an **optimistic** assumption for the relaxed plan cost
  - $\mathbf{h_{add}}$ provides a **pessimistic** assumption for the relaxed plan cost
  - $h_{max} \leq h^{+} \leq h_{add}$

# Heuristic $h_{add}$

$$h_{add}(s)=h_{add}(G;s)$$

$$h_{add}(P;s) = \Sigma_{p\in P}\ h_{add}(p;s)$$

$$h_{add}(p;s) = 0, \text{ if } p\in s$$

$$= a_p(s), \text{ otherwise}$$

$$a_p(s)=\min_{a\in\{a'|p\in add(a')\}} h_{add}(a;s)$$

$$h_{add}(a;s)=c(a)+h_{add}(pre(a);s)$$

Note that s is a state, p an atom, a an action, G a goal and P a set of atoms

# Heuristic $h_{max}$

$$h_{max}(s)=h_{max}(G;s)$$

$$h_{max}(P;s) = \max_{p \in P} h_{max}(p;s)$$

$$h_{max}(p;s) = 0, \text{ if } p \in s$$

$$= a_p(s), \text{ otherwise}$$

$$a_p(s)=\min_{a \in \{a' | p \in add(a')\}} h_{max}(a;s)$$

$$h_{max}(a;s)=c(a)+h_{max}(pre(a);s)$$

Note that s is a state, p an atom, a an action, G a goal and P a set of atoms

# Computation

- Basic idea – **value iteration**
- Set values of initial atoms to 0, and ∞ to other atoms and actions
- If a value of an atom changes, update the values of actions having it in precondition accordingly
- **Label-correcting** action selection method
  - select an **arbitrary** action to process (update the values of atoms in its add effects accordingly)
  - multiple updates per atom
- **Dijkstra** action selection method
  - select the **cheapest** action to process (update the values of atoms in its add effects accordingly)
  - single update per atom

# Reachability graph

- Also known as relaxed planning graph
- Consists of alternating layers of atoms and actions $P_0, A_0, P_1, A_1, \ldots$

$$P_0 = I$$

$$A_i = \{a \mid pre(a) \subseteq P_i\}$$

$$P_{i+1} = P_i \cup \bigcup_{a \in A_i} add(a)$$

- Terminate when $G \subseteq P_i$ or $P_{i+1} = P_i$

# Running Example (relaxed planning task)

$P = \{a,b,c,d,e,f,g,h\}$

$I = \{a\}$

$G = \{c,d,e,f,g\}$

$a_1=(\{a\},\{b,c\})$

$a_2=(\{a,c\},\{d\})$

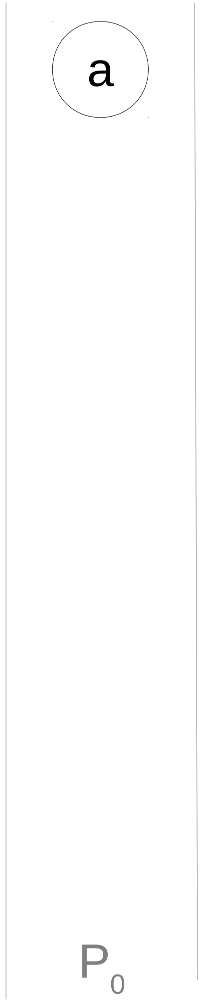
$a_3=(\{b,c\},\{e\})$

$a_4=(\{b\},\{f\})$

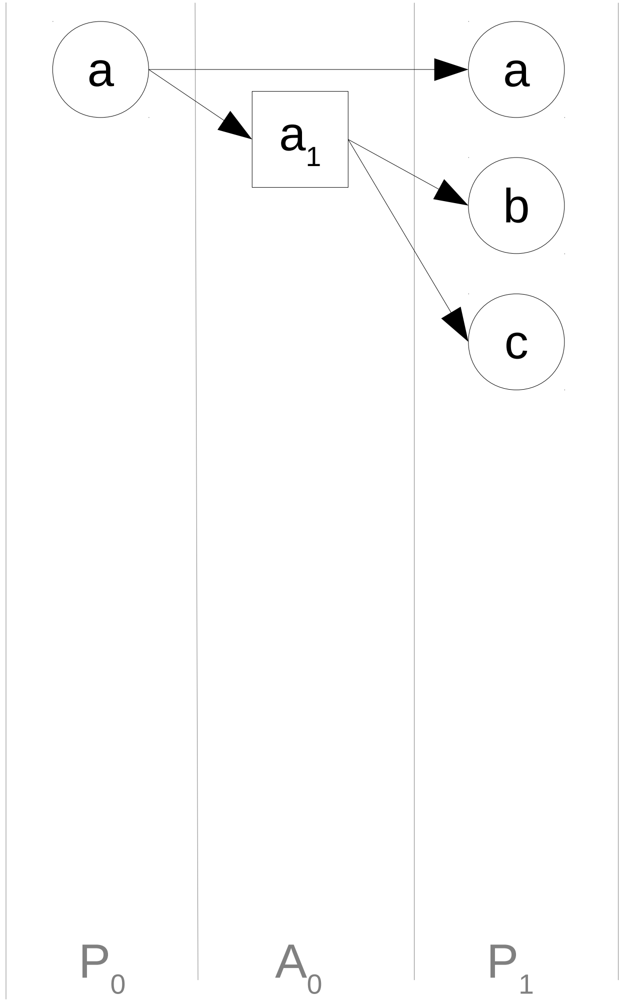
$a_5=(\{d\},\{e,f\})$

$a_6=(\{d\},\{g\})$

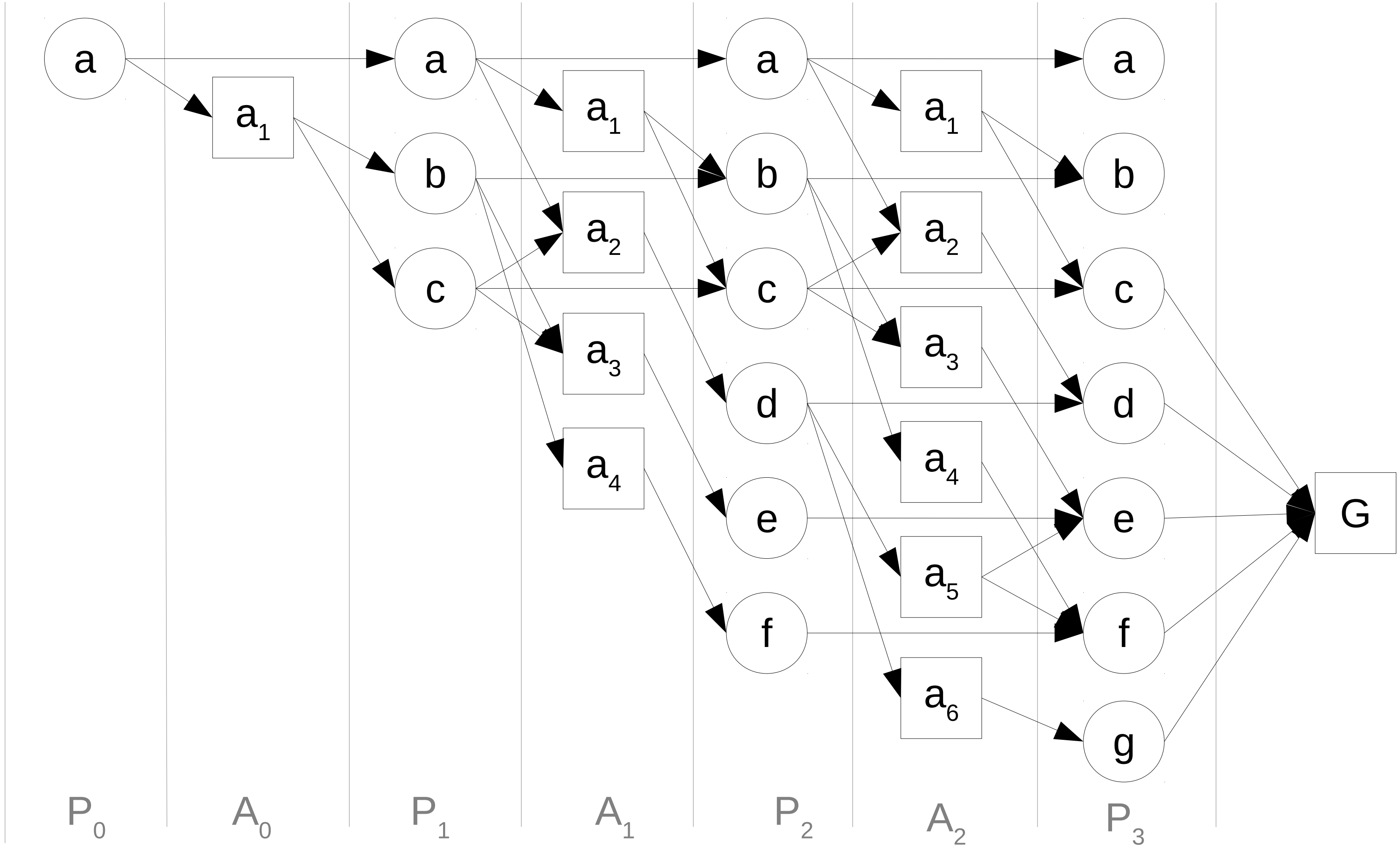
# Running Example: Reachability Graph



# Running Example: Reachability Graph



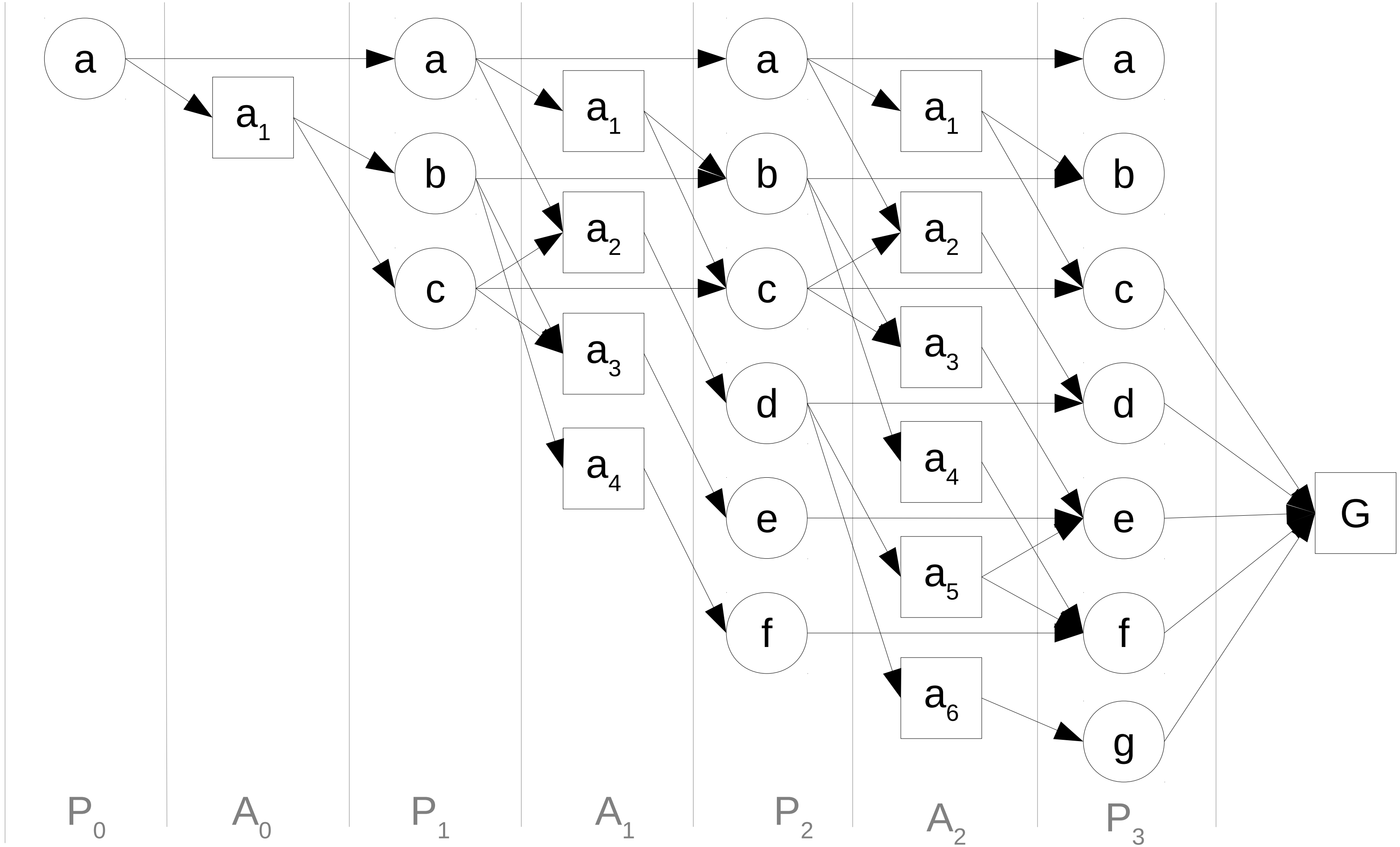
# Running Example: Reachability Graph



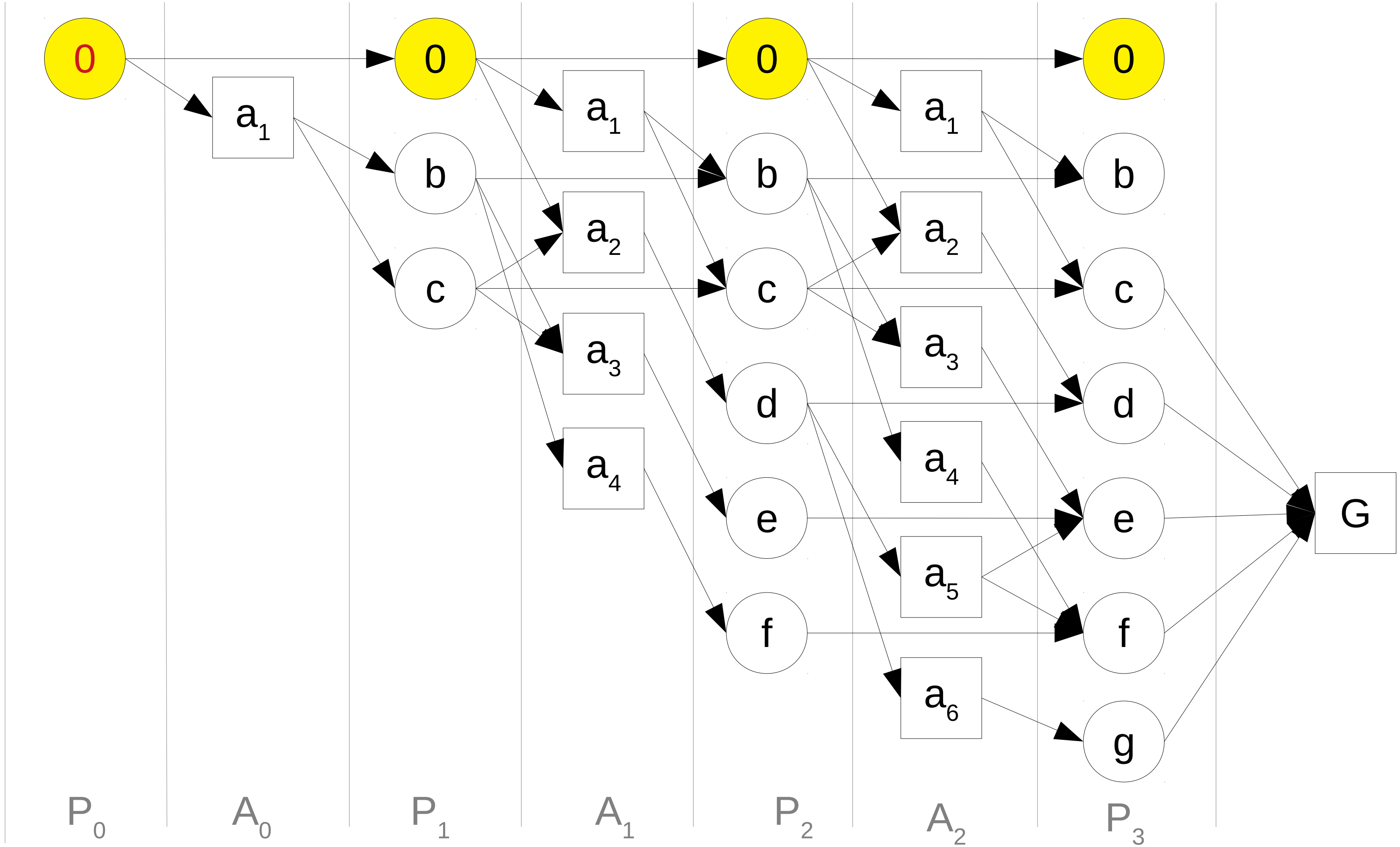
# Using Reachability Graph for computing $h_{max}$ and $h_{add}$

- For **uniform cost** planning tasks we can leverage reachability graph
  - It's a special case of the **Dijkstra action selection method**
- Initially, the reachability graph is constructed from I (or any state s)
  - If a fixed point is reached, i.e., $P_{i+1}=P_i$, then $h_{max}(I)=h_{add}(I)=\infty$
- Then actions are processes layer by layer (from $A_0$, $A_1$, …) until G is reached
  - The value in G is the value of the heuristic for I (or s)

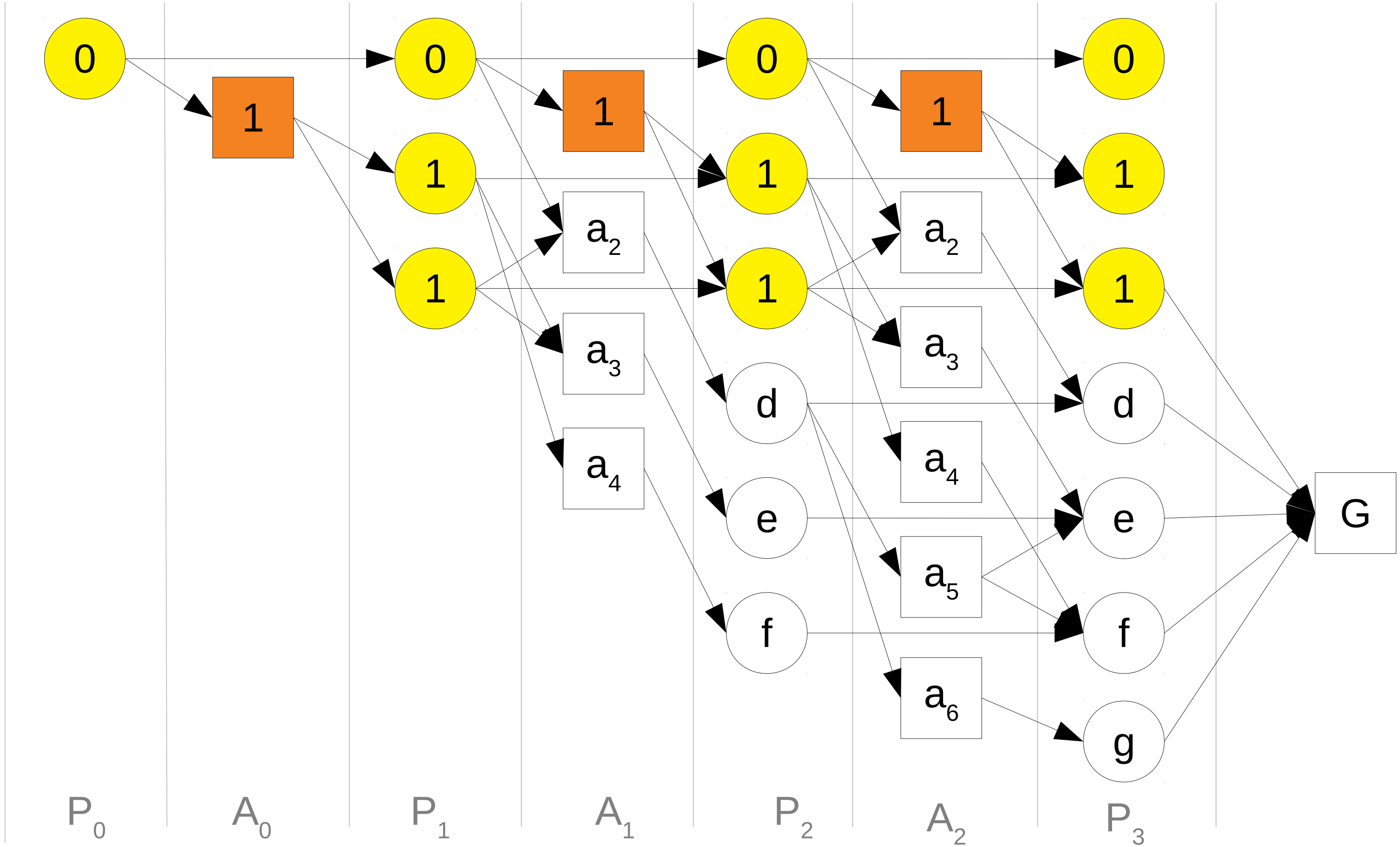
# Running Example: $h_{max}$



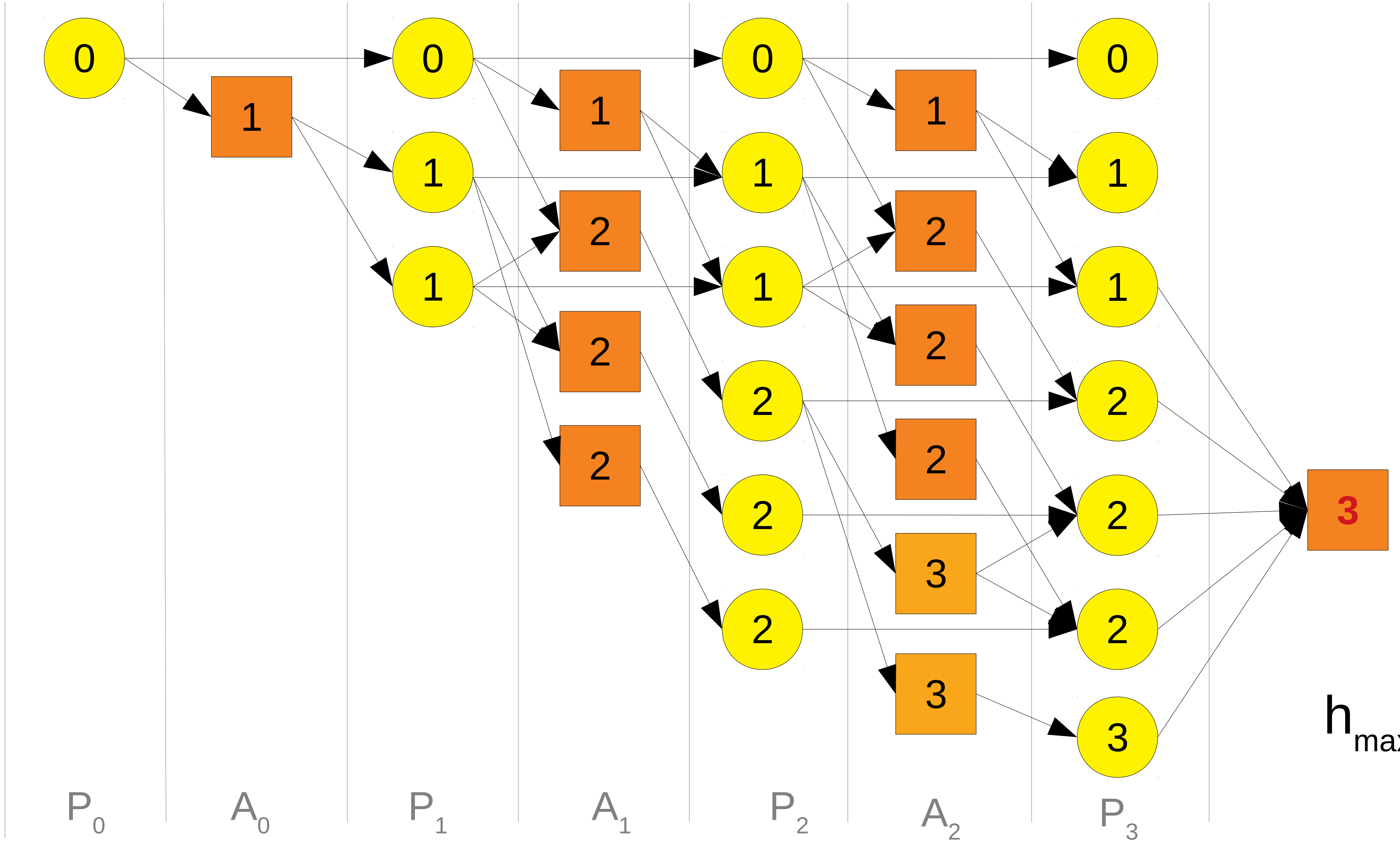
# Running Example: $h_{max}$



# Running Example: $h_{max}$

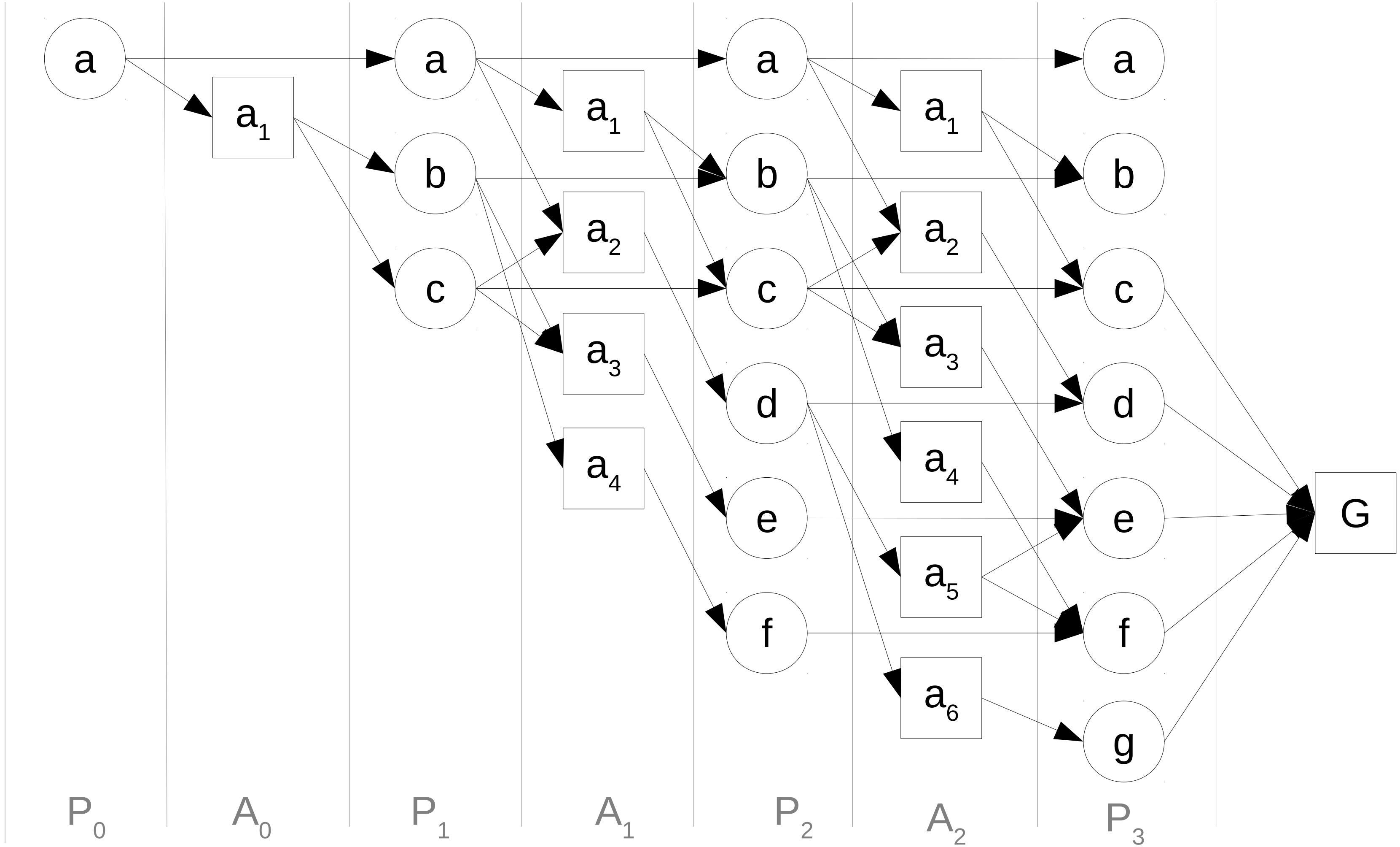


# Running Example: $h_{max}$

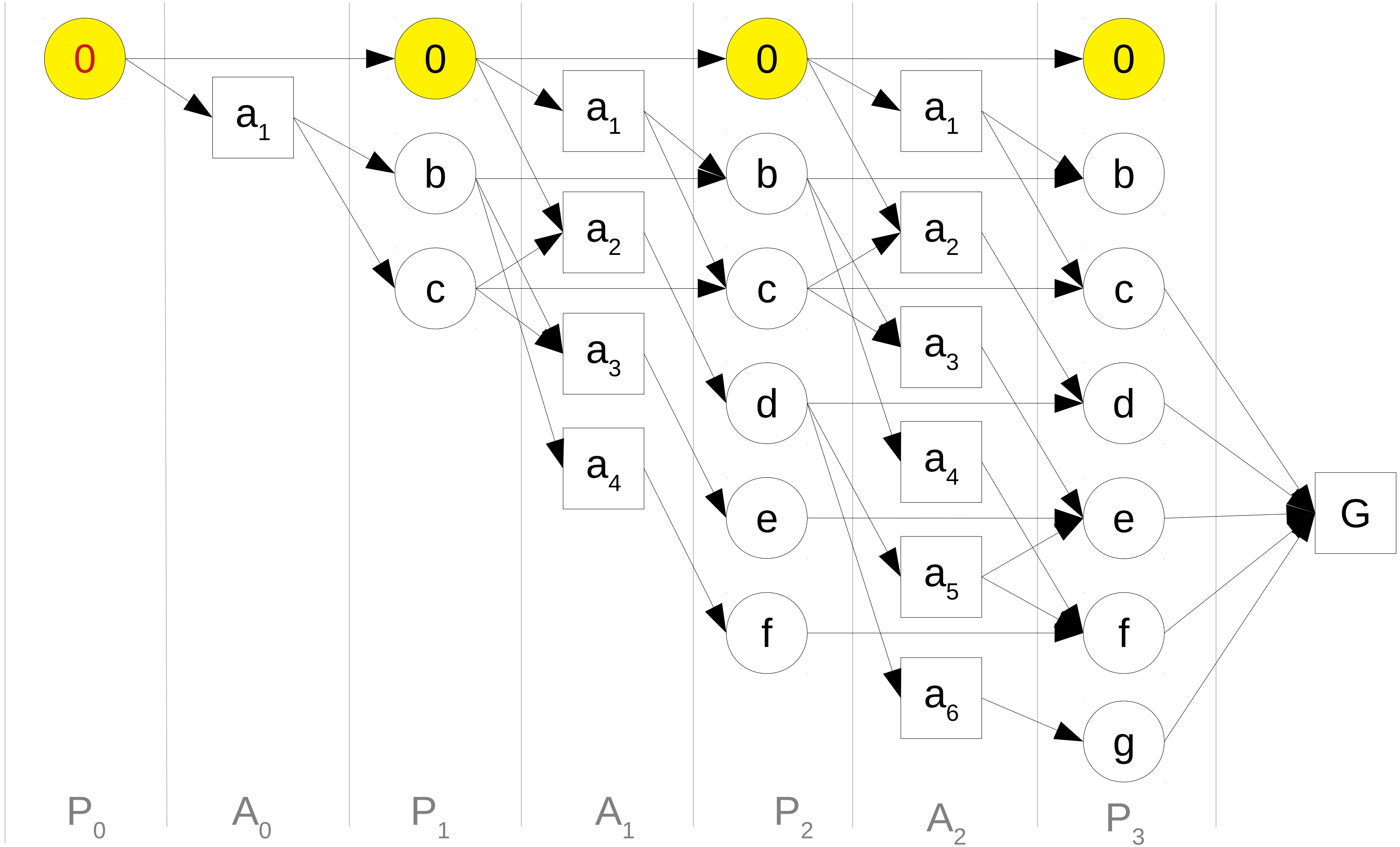



$h_{max}(I)=3$

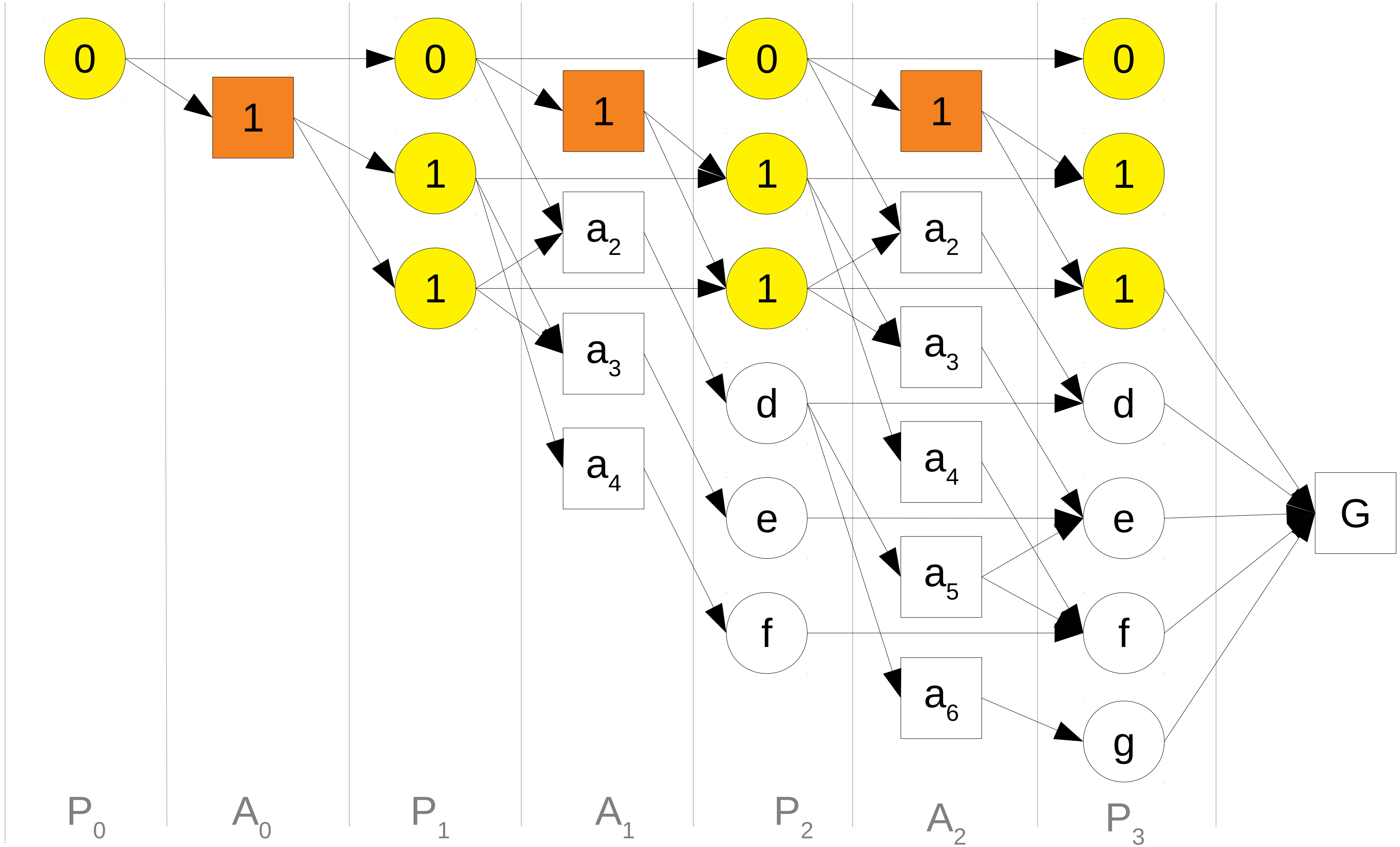
# Running Example: $h_{add}$



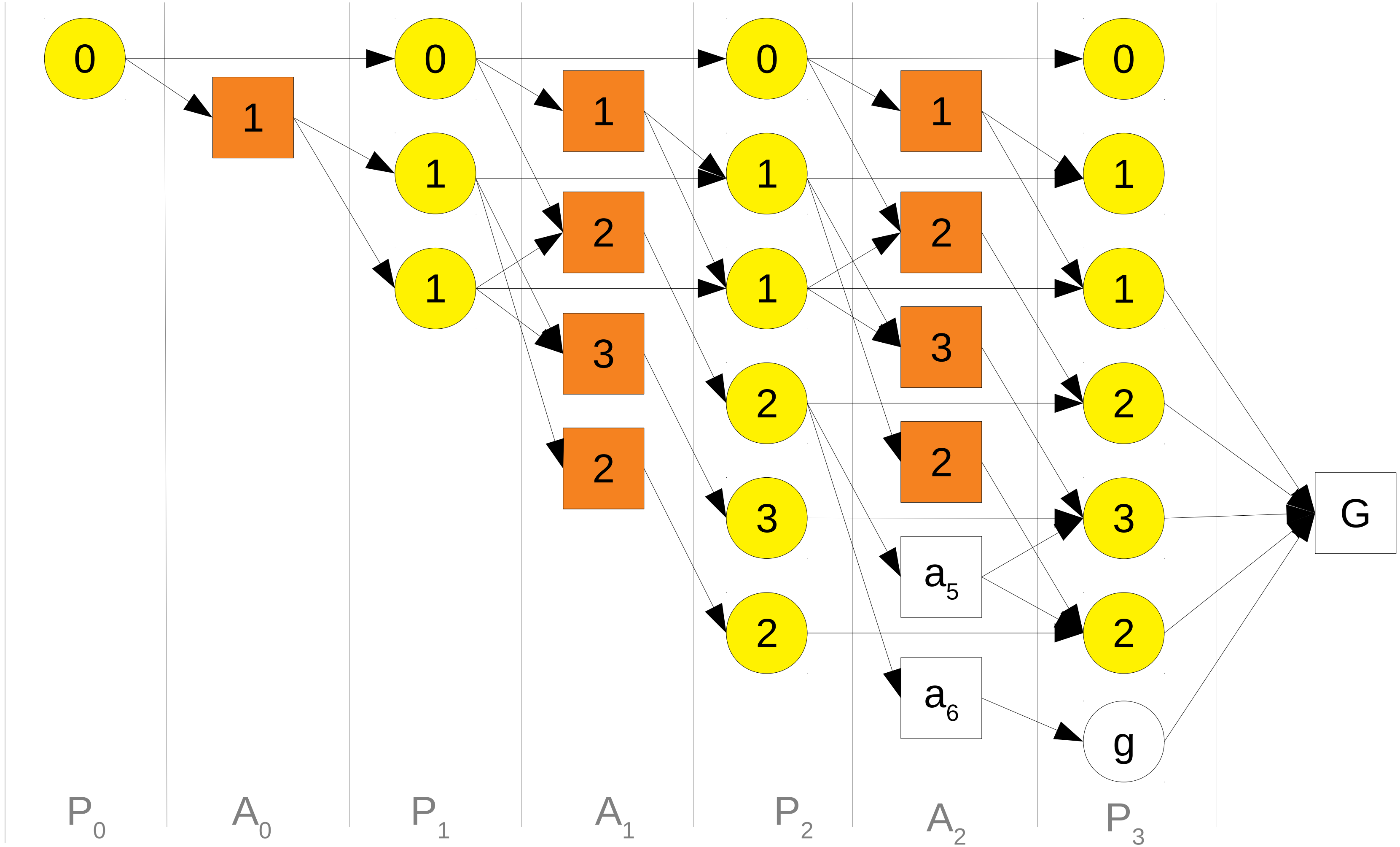
# Running Example: $h_{add}$



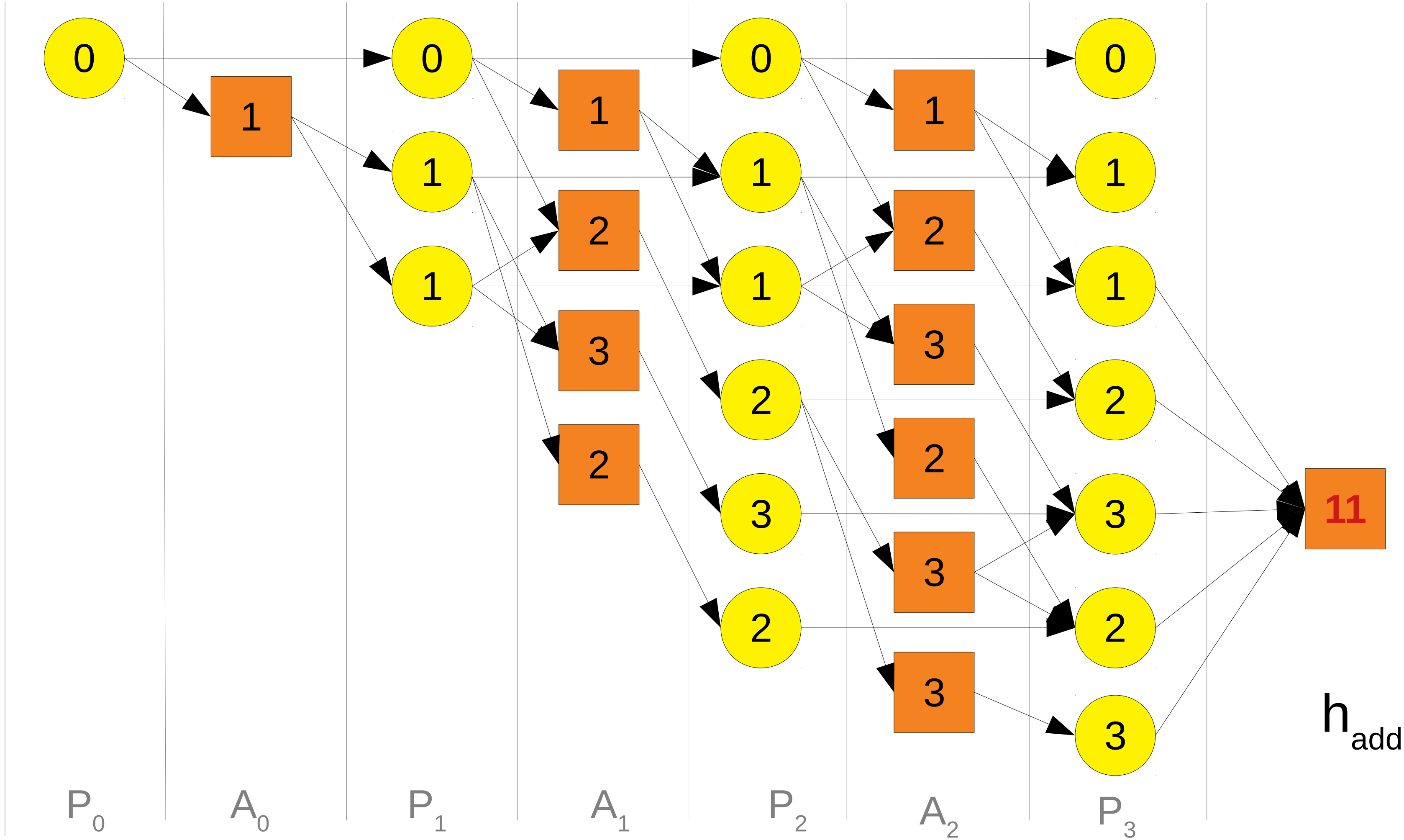
# Running Example: $h_{add}$



# Running Example: $h_{add}$



# Running Example: $h_{add}$




$h_{add}(I)=11$

# Remarks

- $h_{max}$ is sometimes too optimistic as it assumes that some (parallel) actions count as one
  - e.g. loading and unloading multiple packages into/from the truck
- $h_{add}$ is sometimes too pessimistic as it assumes that each atom is achieved by a separate process
  - e.g. moving a block from a tower can both place the block in the right place and clears the block underneath
- Generally, $h_{add}$ is more informative than $h_{max}$ albeit being inadmissible

# $h_{FF}$

- Generates whole relaxed plans (suboptimal but often reasonable)
- Reachability graph is initially generated and the **goal node is marked**
  - If, however, a fixed point is reached, i.e., $P_{i+1}=P_i$, then $h_{FF}(I)=\infty$
- Each action or atom node can be either **marked** or **unmarked**
- A **marked action node** is **justified** if all its predecessors (atom nodes) are **marked**
- A **marked atom node** is **justified** if at least one of its predecessors is **marked**

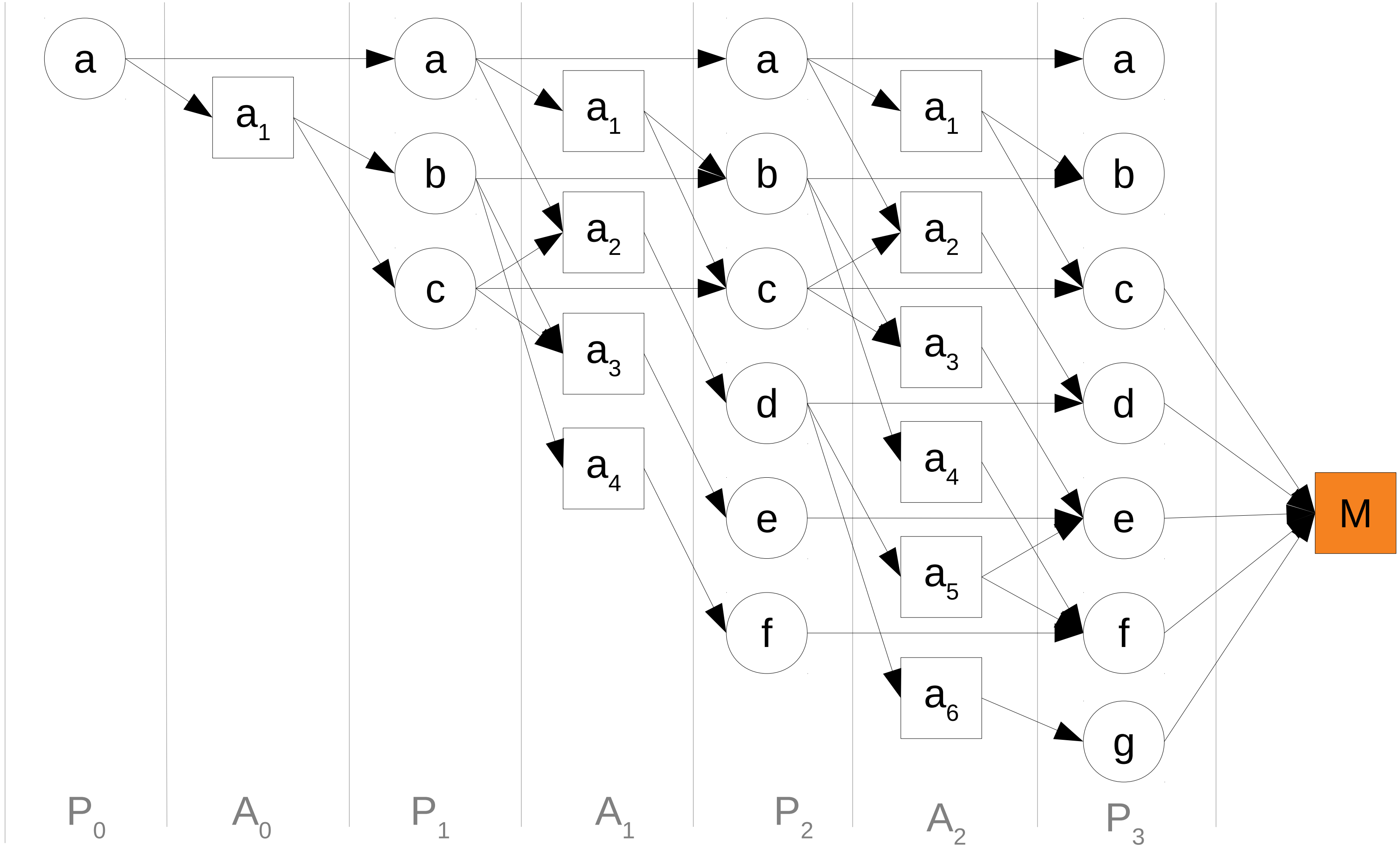
# $h_{FF}$

- Starting with a **marked goal node**, apply the following rules until **all marked nodes are justified**

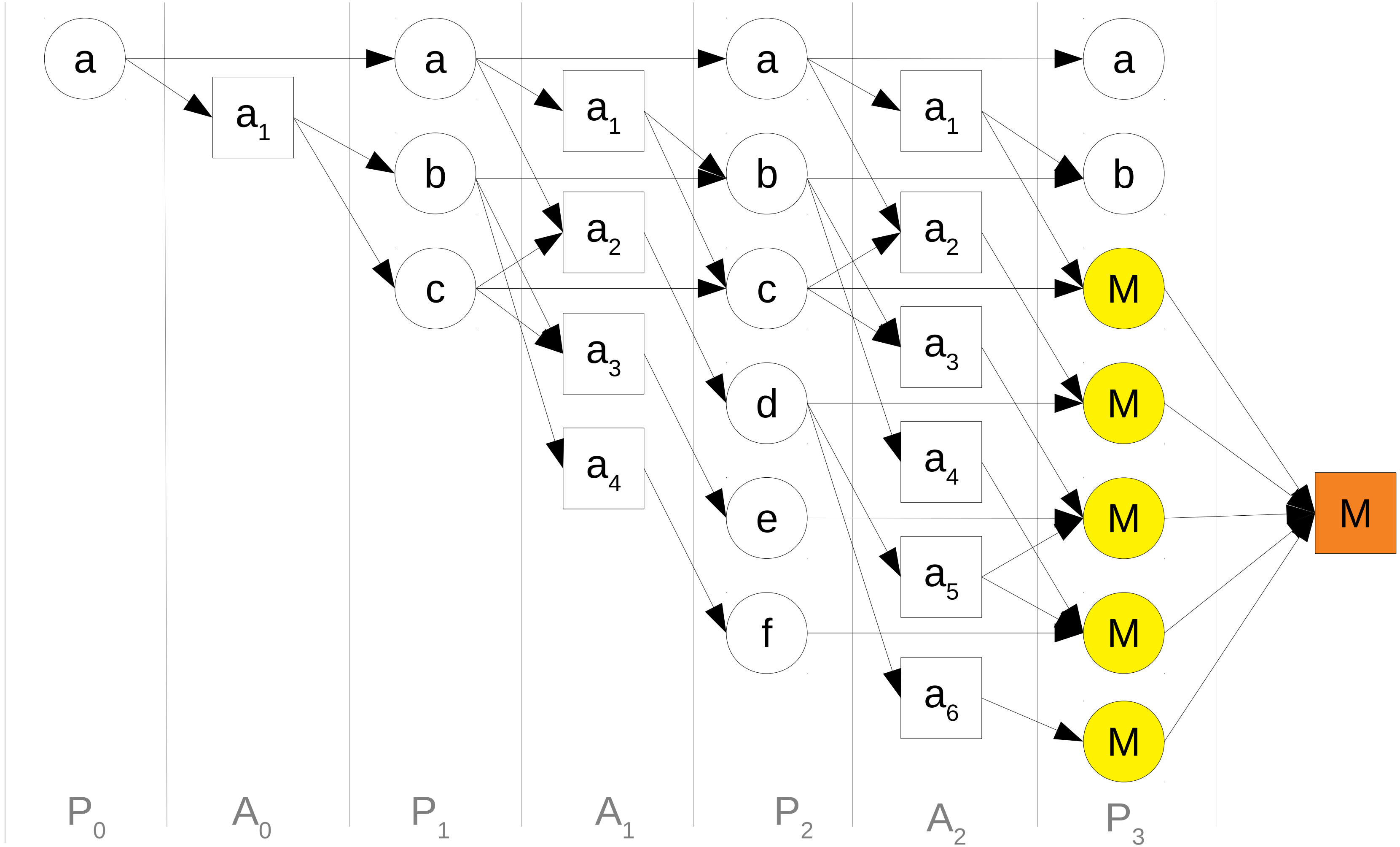
1) Mark all immediate predecessors of a marked unjustified action node
2) Mark the immediate predecessor of a marked unjustified atom node with only one immediate predecessor
3) Mark an immediate predecessor of a marked unjustified atom node connected via an idle arc (to the same atom in the previous layer)
4) Mark any immediate predecessor of a marked unjustified atom node

- The rules are applied in a **priority order** (earlier first if applicable)
- The **number** (or the **total cost**) of **marked action nodes** is the $\mathbf{h_{FF}}$ **value**

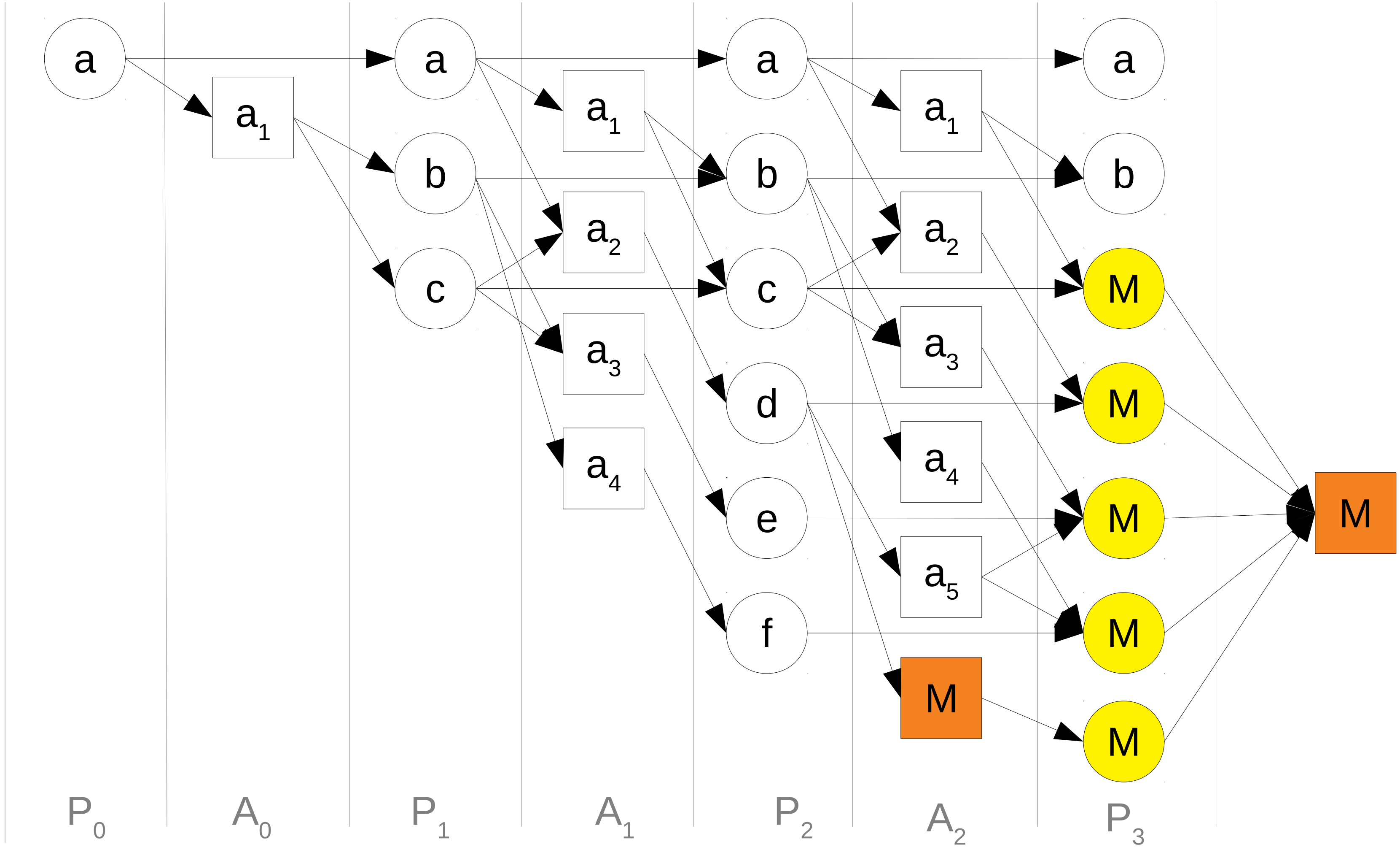
# Running Example: $h_{FF}$



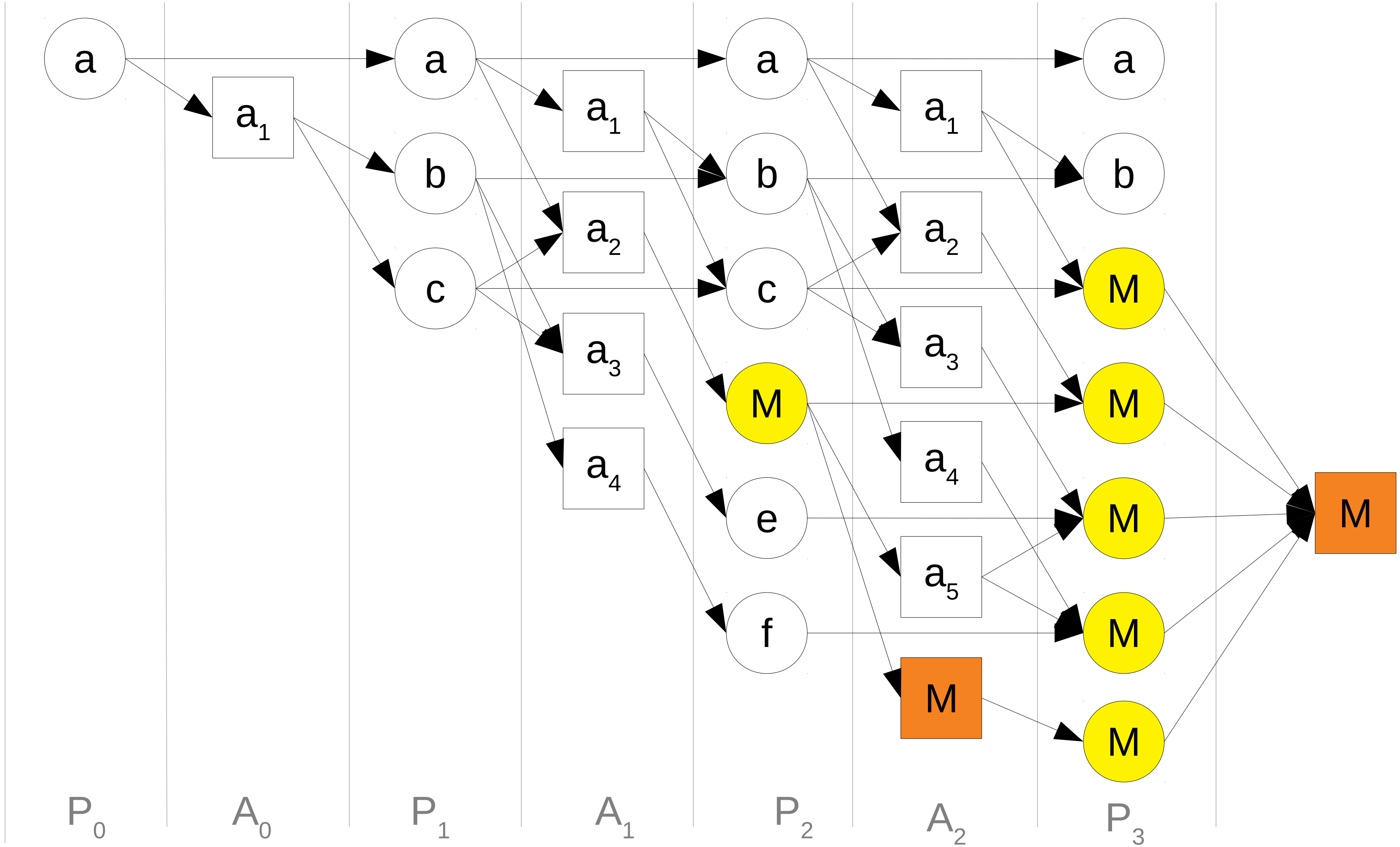
# Running Example: $h_{FF}$



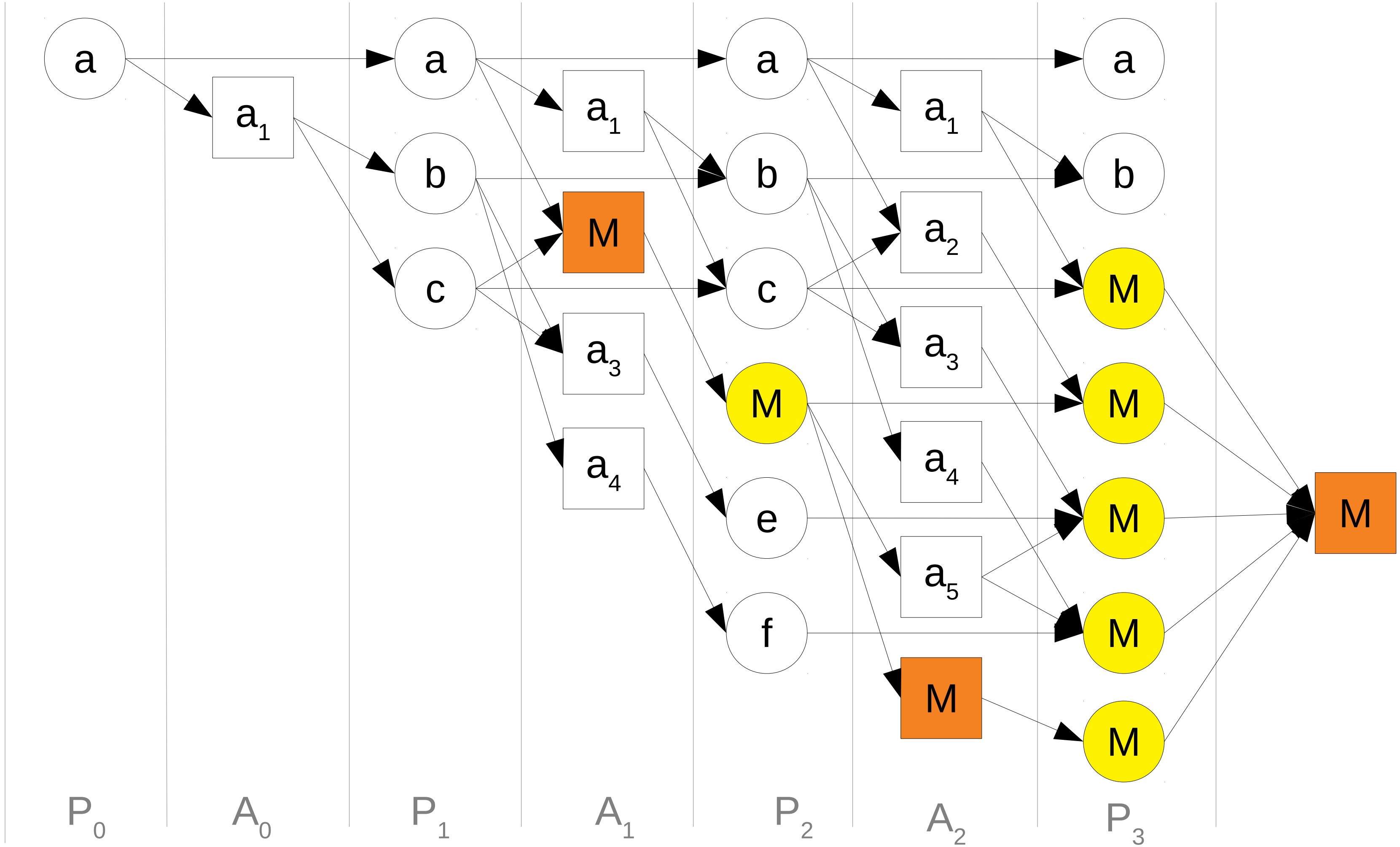
# Running Example: $h_{FF}$



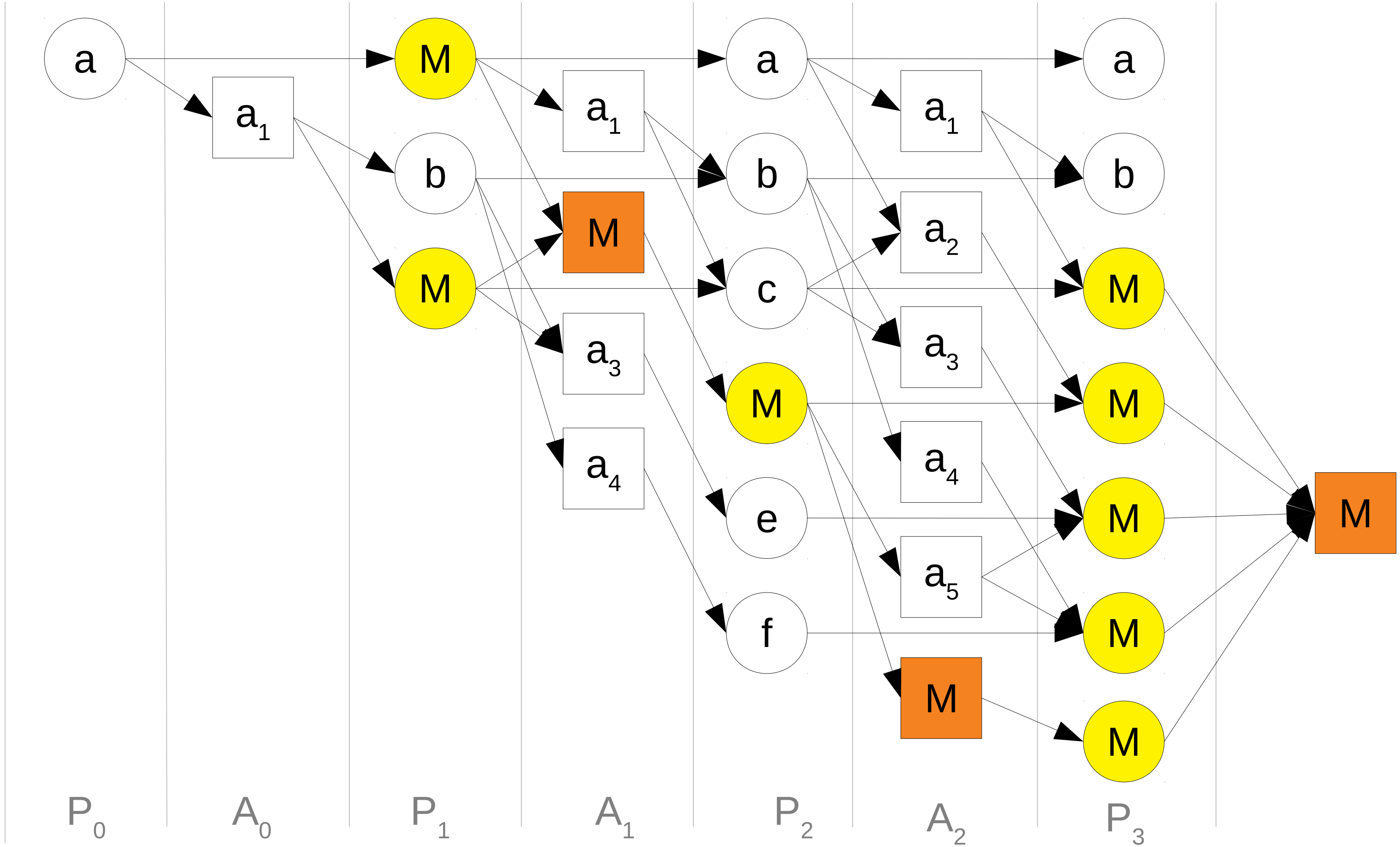
# Running Example: $h_{FF}$



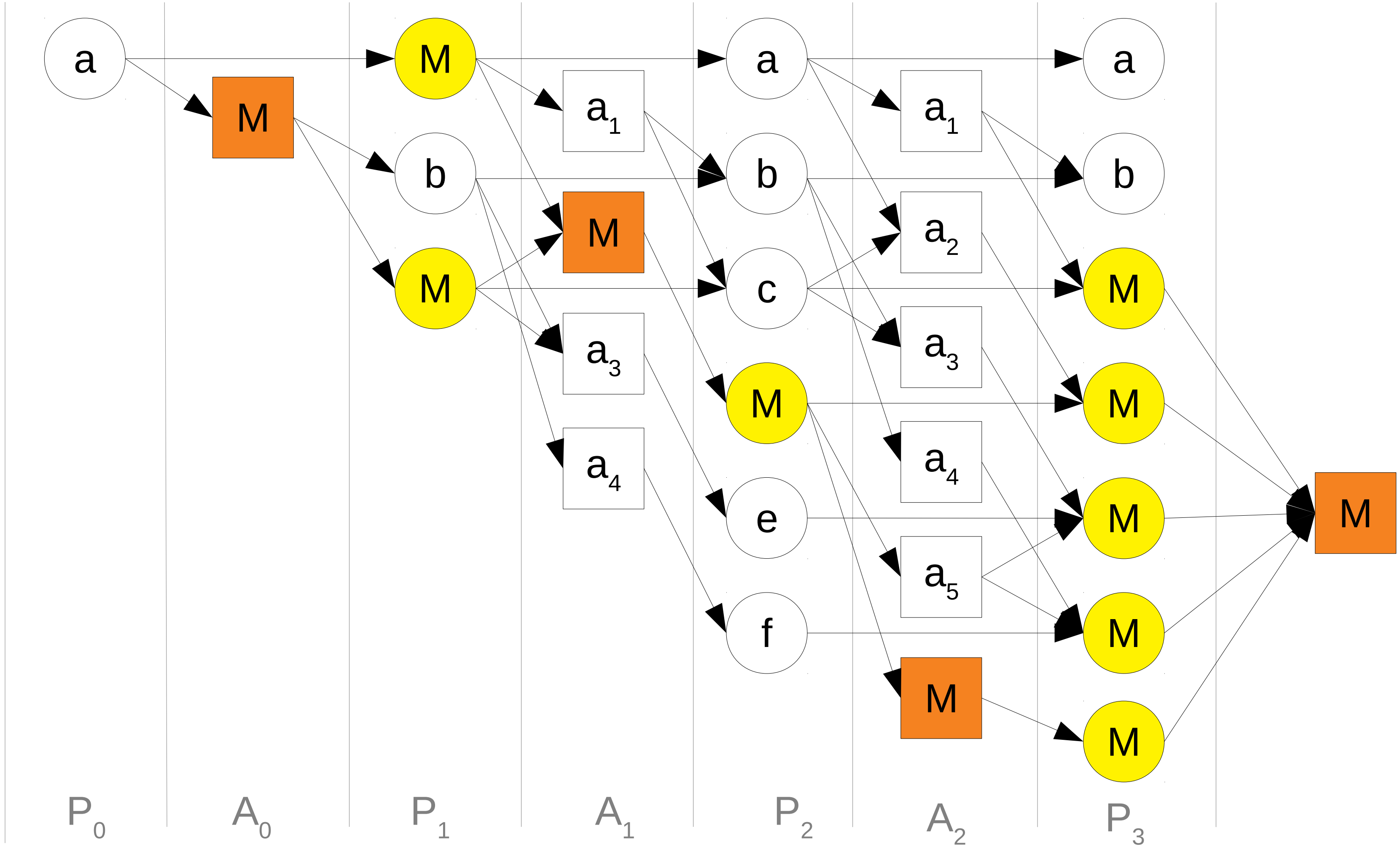
# Running Example: $h_{FF}$



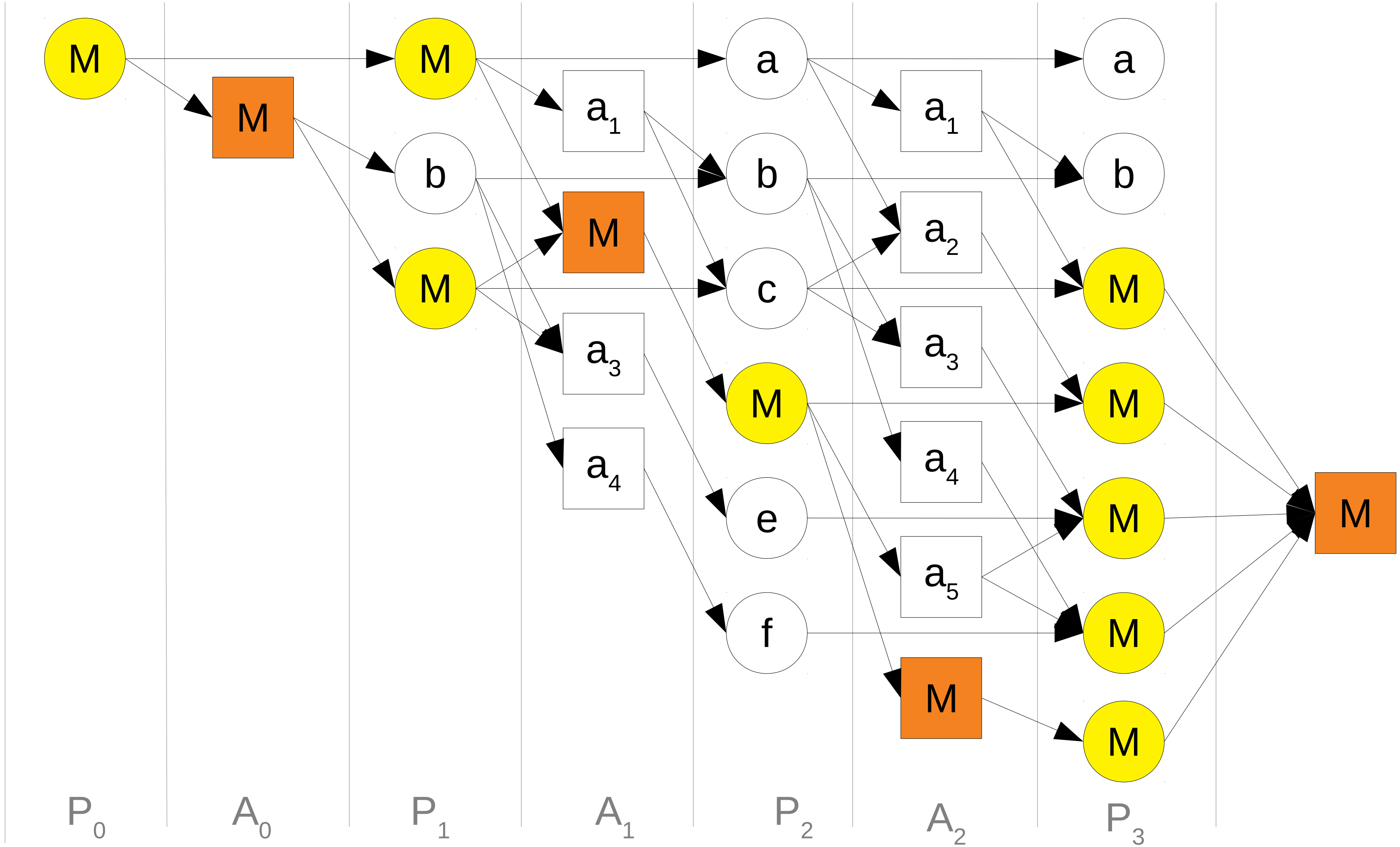
# Running Example: $h_{FF}$



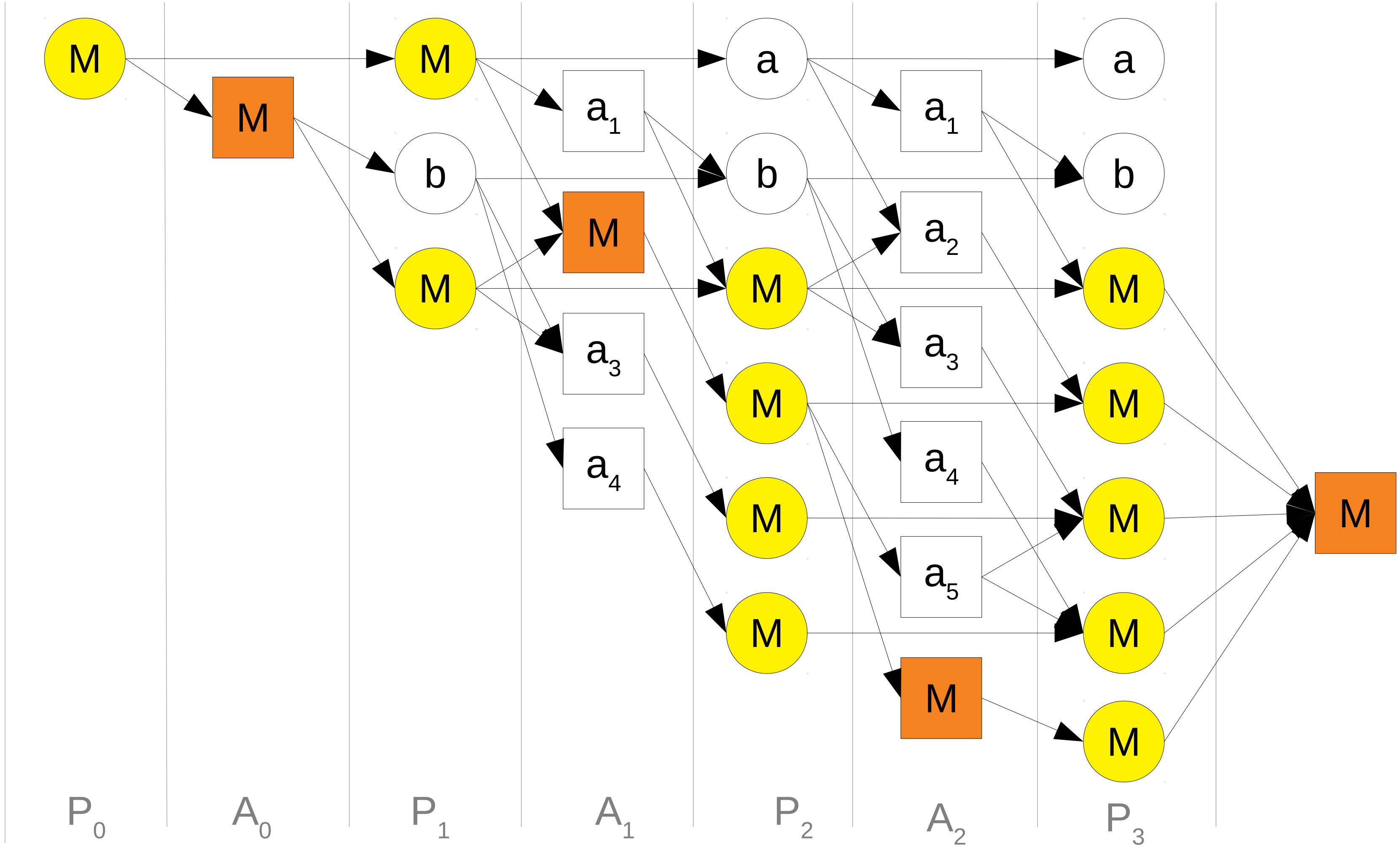
# Running Example: $h_{FF}$



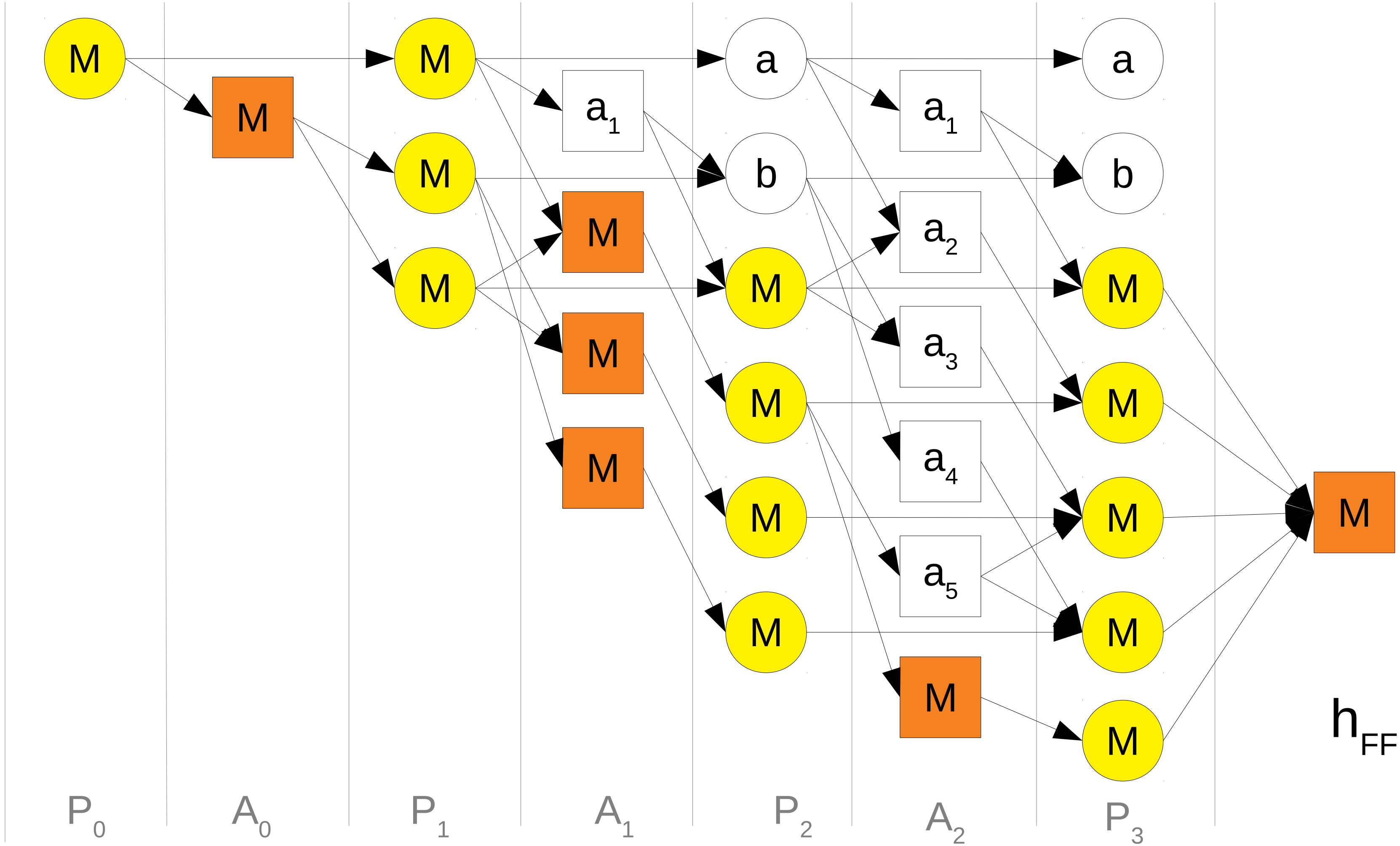
# Running Example: $h_{FF}$



# Running Example: $h_{FF}$



# Running Example: $h_{FF}$




$h_{FF}(I)=5$

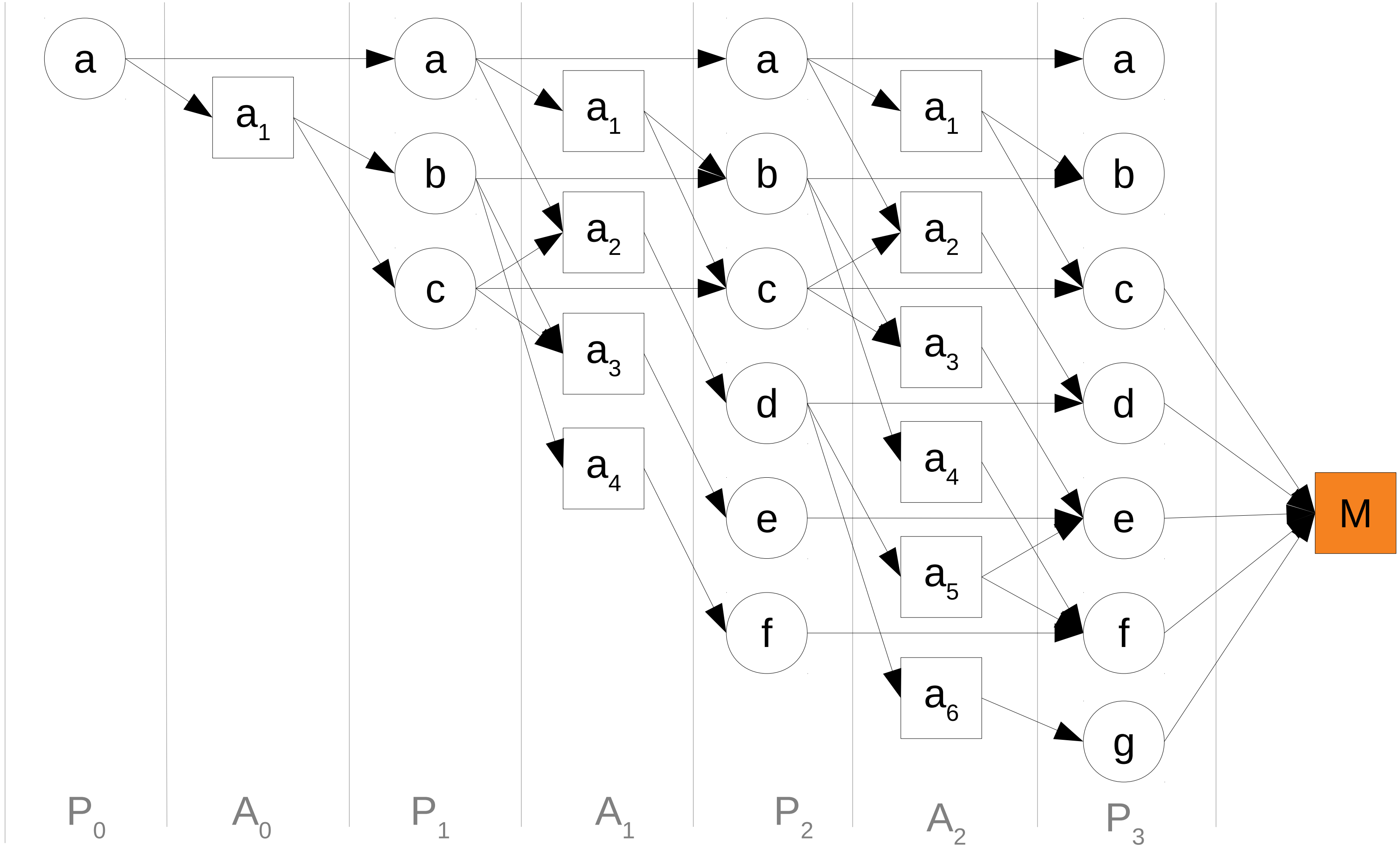
# $h_{FF}$ (layer by layer)

- Starting with marked goal node, apply the following rules **layer by layer** until **all marked nodes are justified**

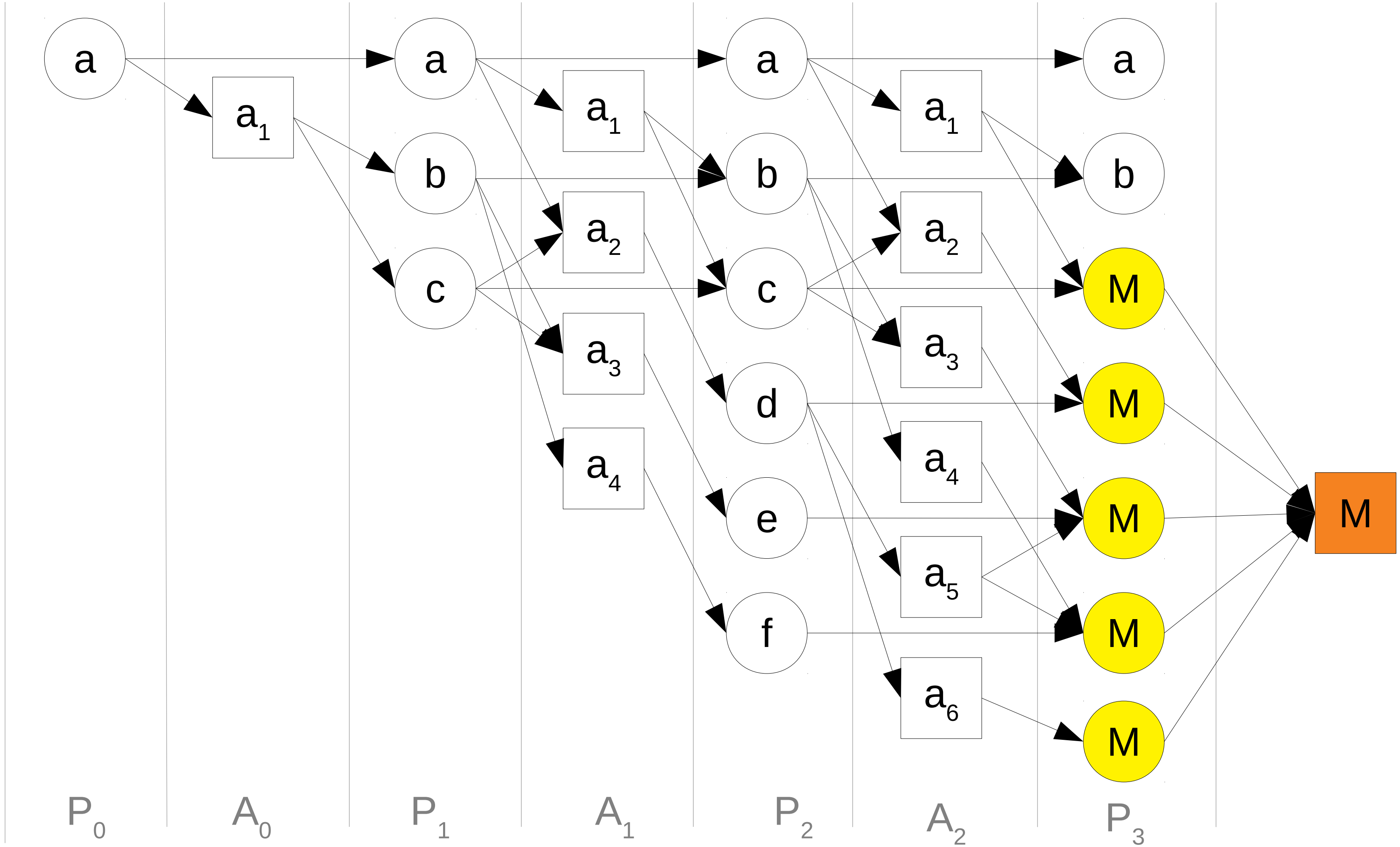
1) Mark all immediate predecessors of a marked unjustified action node
2) Mark the immediate predecessor of a marked unjustified atom node with only one immediate predecessor
3) Mark an immediate predecessor of a marked unjustified atom node connected via an idle arc (to the same atom in the previous layer)
4) Mark any immediate predecessor of a marked unjustified atom node

- The rules are applied in a **priority order** (earlier first if applicable)
- The **number** (or the **total cost**) of **marked action nodes** is the $\mathbf{h_{FF}}$ **value**

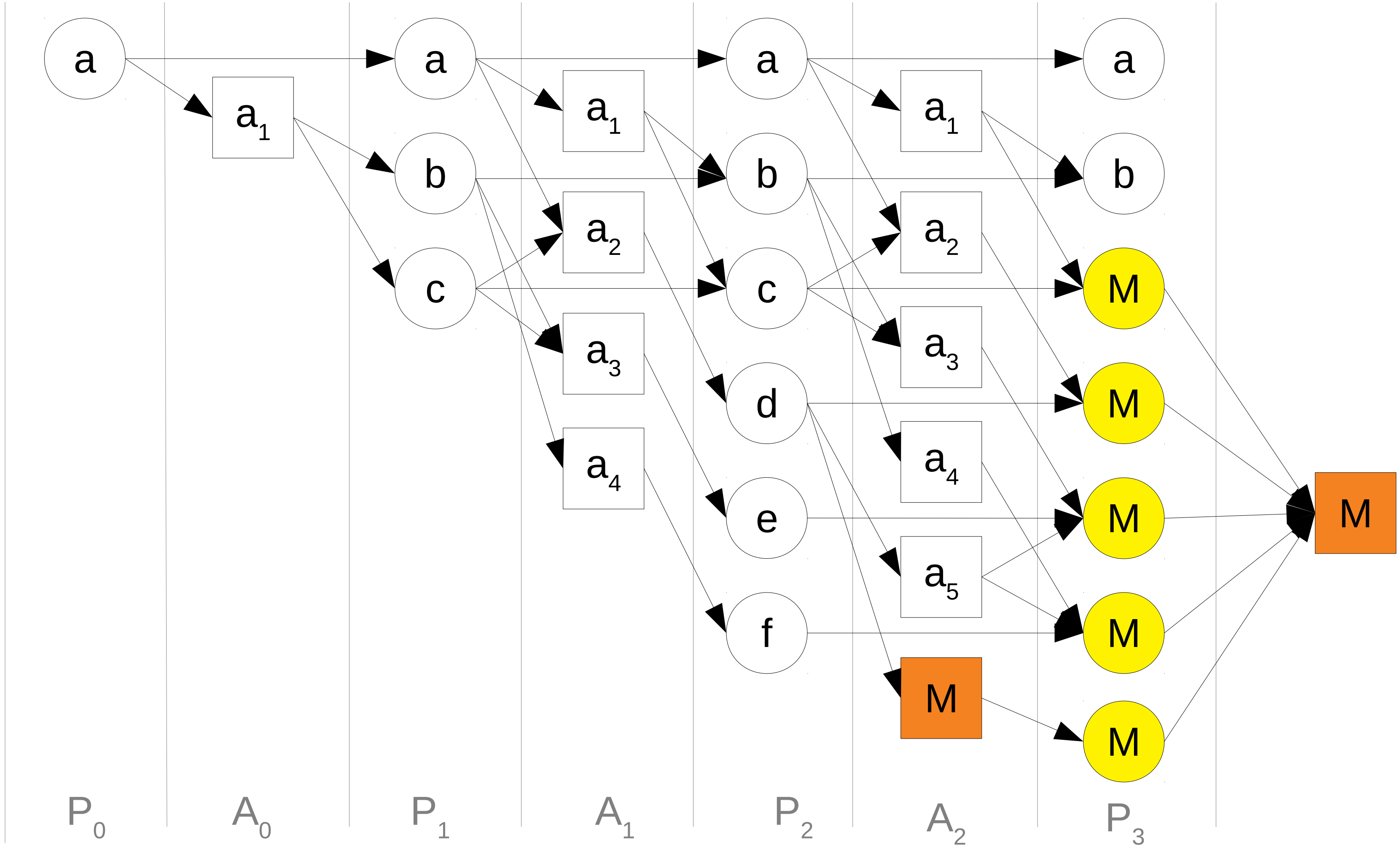
# Running Example: $h_{FF}$ (layer by layer)



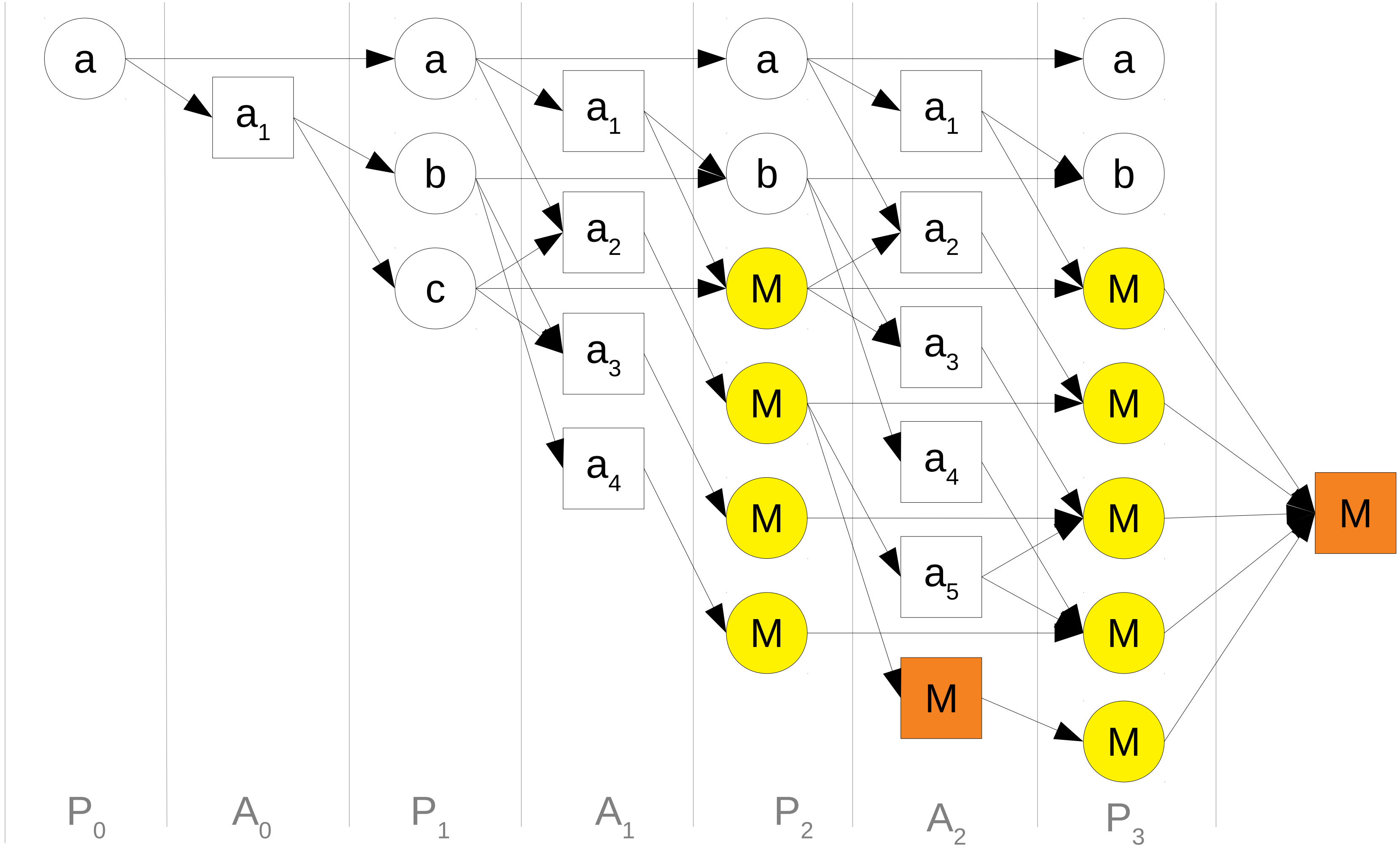
# Running Example: $h_{FF}$ (layer by layer)



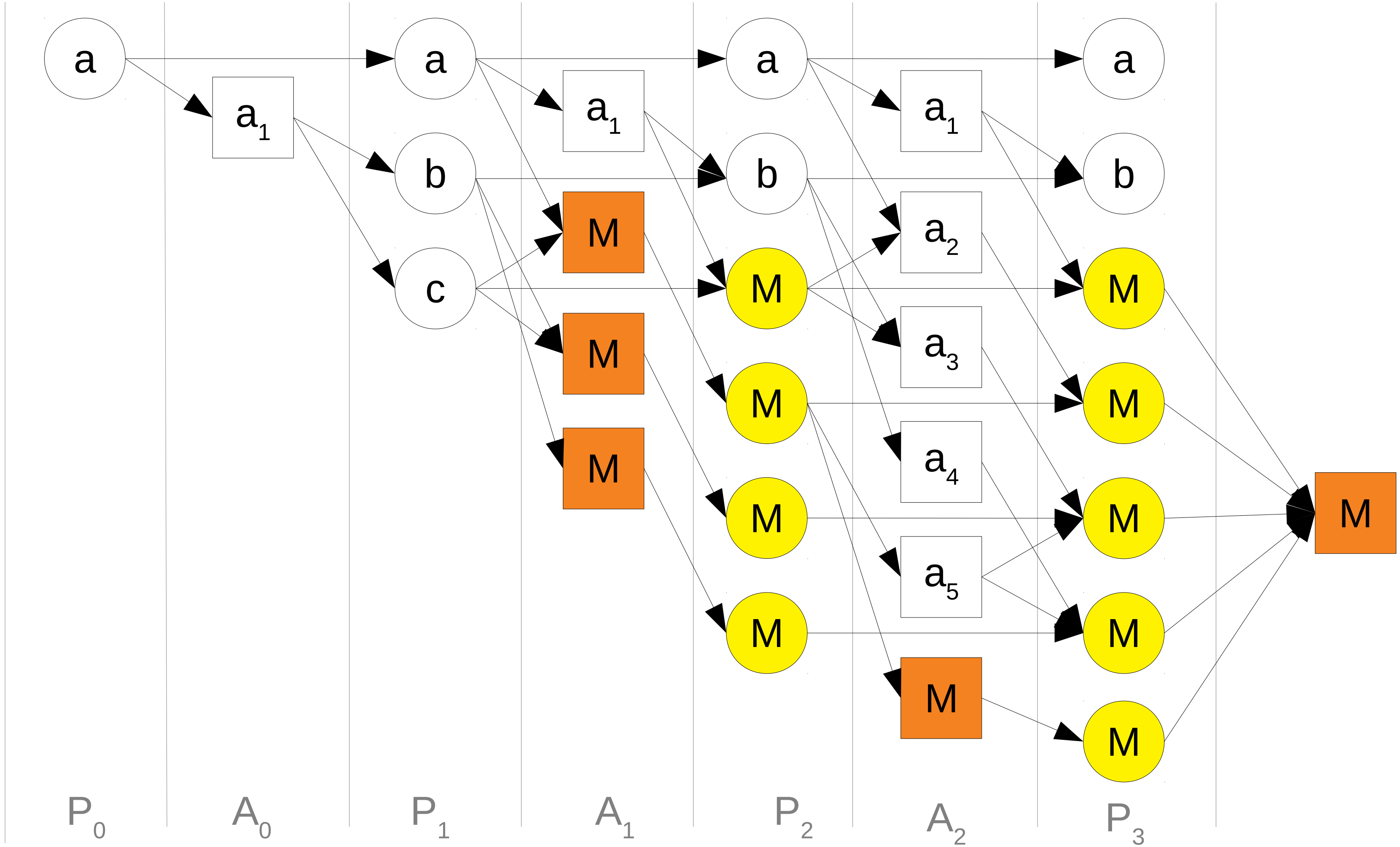
# Running Example: $h_{FF}$ (layer by layer)



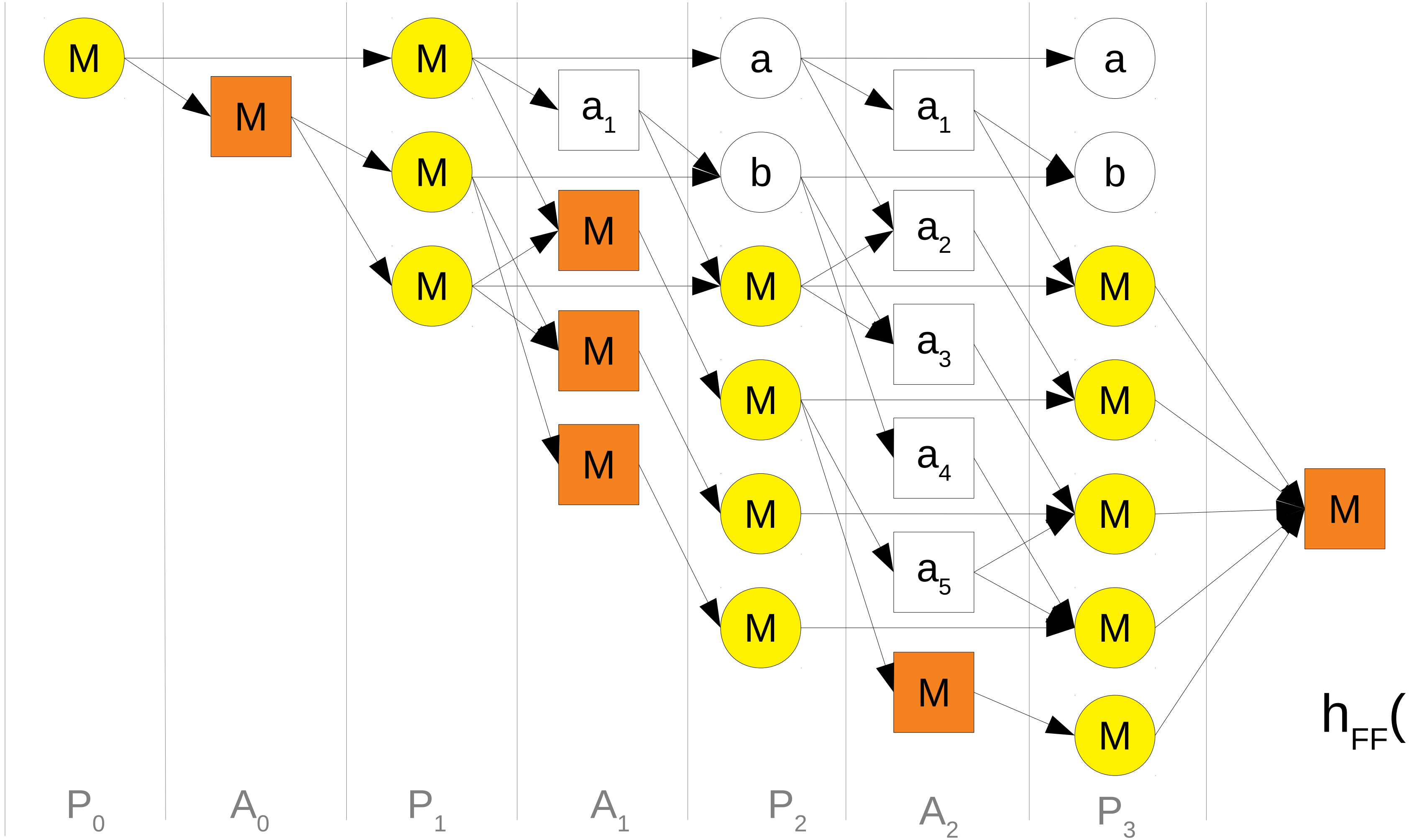
# Running Example: $h_{FF}$ (layer by layer)



# Running Example: $h_{FF}$ (layer by layer)



# Running Example: $h_{FF}$ (layer by layer)




$h_{FF}(I)=5$

# $h_{FF}$ Remarks

- $h_{FF}$ is not well defined as tie-breaking might lead to different values
- $h_{max} \leq h^{+} \leq h_{FF} \leq h_{add}$
- FF planner won the second IPC (in 2000)
- Note that delete-relaxation has some drawbacks (e.g. some non-detected dead-ends)