Základy umělé inteligence

Automatické plánování

Ing. Tomáš Řehořek

Katedra teoretické informatiky (KTI), Fakulta informačních technologií (FIT)
České vysoké učení technické v Praze (ČVUT)

BI-ZUM, LS 2016/17, 6. přednáška

https://edux.fit.cvut.cz/courses/BI-ZUM/

Heuristiky v AI

- Viděli jsme, že spoustu problémů AI lze formulovat pomocí **stavového prostoru**,
- Stavový prostor pak můžeme prohledat:
  - **neinformovanými algoritmy** – Random search, BFS, DFS, Dijkstra,
    - ★ použitelné jen pro triviální problémy,
  - **informovanými algoritmy** – Hladové prohledávání, A*,
    - ★ nezbytné pro problémy z praxe,
- Typicky pro daný problém potřebujeme dodat **specializovanou heuristiku**
  - manhattanská vzdálenost pro „Lišáka“ ($N \times N$ puzzle),
  - počet kolizí pro N dam,
  - vzdušná vzdálenost pro hledání cesty na mapě,
- Nutnost dodat heuristiku je **nežádoucí**
  - člověkem vymyšlená heuristika $\neq$ AI,
  - od skutečné AI očekáváme, že bude sama tvořit heuristiky!

General Problem Solving (GPS)

$\neq$ Global Positioning System

- Tradiční řešení problémů v informatice:

  Problém $\longrightarrow$ Člověk $\longrightarrow$ **Tvorba specializovaného programu** $\longrightarrow$ Řešení

- Tento postup má své nevýhody
  - vymyslet a implementovat algoritmus pro daný typ problému je nákladné,
  - optimalizace a ladění specializovaného algoritmu může trvat několik let,
- **General Problem Solving**:

  Problém $\longrightarrow$ Člověk $\longrightarrow$ Zápis v jazyce $\longrightarrow$ **Solver** $\longrightarrow$ Řešení

- Nechť **univerzální solver** koncentruje, vytváří a vybírá heuristiky!
  - po uživateli vyžadujeme pouze formulaci problému

General Problem Solving (GPS)

Obecný princip GPS přesněji:

$$\text{Problém} \longrightarrow \text{Model} \longrightarrow \text{Jazyk} \longrightarrow \text{Solver} \longrightarrow \text{Řešení}$$

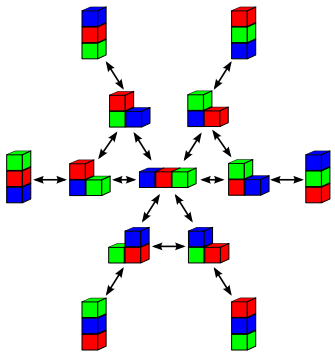
1. uvažuj **problém** P,
2. zvol vhodný **model** M,
3. zvol vhodný **jazyk** L pro zápis modelu M,
4. zvol vhodný **solver** S pro řešení problémů v L,
5. nech **automaticky vyřešit** $S(L(P))$,
6. **interpretuj** $S(L(P))$ v kontextu původního problému P

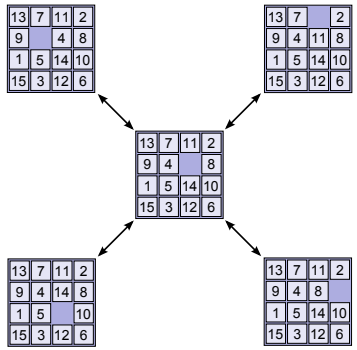
Automatické plánování

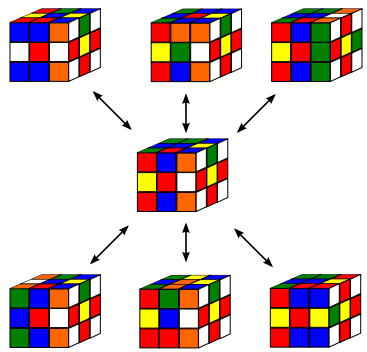
Náplň dnešní přednášky: **Automatické plánování**

- „Lišák“ ($N \times N$ puzzle),
- Rubikova kostka,
- Sekvence akcí jeřábu pro přesun kontejnerů,
- Návaznosti akcí na výrobních linkách,

Všechny tyto problémy dokážeme snadno formulovat pomocí stavového prostoru, kde každý stav představuje jistou **strukturovanou konfiguraci** a akce provádějí změny struktury konkrétních konfigurací







Plánovací modely

- Dosud jsme při řešení problémů pohlíželi na stavový prostor **„staticky“**:
  - stavový prostor je kompletně zadán výčtem stavů a akcí,
  - graf prostoru máme „nakreslen“ a stačí jej prohledat,
- Praktické problémy vyžadují **„dynamickou“** optiku:
  - předem znám je pouze **počáteční stav**, v něm hledání začínáme,
  - prostor prohledáváme **generováním** následníků dosud objevených stavů,
  - to nám umožňuje **kompaktní zápis** problému
- V dnešní přednášce se nadále budeme zabývat speciální třídou stavových prostorů, které lze kompaktně vyjádřit jako **plánovací úlohy**
- Ukážeme si
  1. **model** jednoduchého plánovacícho problému,
  2. **jazyk** pro zápis tohoto modelu,
  3. **algoritmy** pro řešení problémů v tomto jazyce

Plánovací modely

Nabízí se široká škála možných plánovacích modelů:

- **Dynamika systému**
  - **deterministická** – akce mají jediný výsledek, jsou vždy „úspěšné“,
  - **nedeterministická** – akce mají více možných výsledků,
  - **pravděpodobnostní** – akce mají více možných výsledků popsaných pravděpodobnostmi,
- **Pozorovatelnost**
  - **úplná** – v každém okamžiku můžeme pozorovat kompletní konfiguraci problému,
  - **částečná** – konfiguraci v každém stavu pozorujeme jen částečně (např. kamera pohlížející na kontejnery shora),
  - **žádná** – nemáme zpětnou vazbu o akcích, které provádíme
- **Spojitost**
  - **diskrétní stavový prostor** – „skokový“ přesun mezi stavy,
    - **konečně** mnoho stavů,
    - **spočetně nekonečně** mnoho stavů,
  - **spojitý stavový prostor** – „plynulý“ přesun mezi stavy,
    - **nespočetně** mnoho stavů

Plánovací modely

Podle toho, jaké vlastnosti problém má, rozlišujeme různé plánovací modely:

- **klasické plánování**,
- podmíněné plánování s úplnou pozorovatelností,
- podmíněné plánování s částečnou pozorovatelností,
- Markovovské rozhodovací procesy,
- Markovovské rozhodovací procesy s částečnou pozorovatelností

V dnešní přednášce se budeme zabývat pouze **klasickým plánováním**:

- deterministická dynamika,
- plně pozorovatelný počáteční stav,
  - akce jsou deterministické, tedy pozorovatelnost v dalších stavech nehraje roli,
- konečný stavový prostor

Klasické plánování v jazyku STRIPS

Definice: Plánovací úloha ve STRIPS

Problém ve STRIPS je čtveřice $(P, A, \mathcal{I}, G)$, kde

- P je konečná množina **atomů**,
- A je konečná množina **akcí** popsaných pomocí $\text{pre}(a) \subseteq P$, $\text{add}(a) \subseteq P$, $\text{del}(a) \subseteq P$,
- $\mathcal{I} \subseteq P$ je **počáteční situace**,
- $G \subseteq P$ je **koncová situace**.

Plánovací úloha $\neq$ stavový prostor,

- Plánovací problém = kompaktní forma zápisu stavového prostoru

Jazyk STRIPS

Úloha $(P, A, \mathcal{I}, G)$ ve STRIPS indukuje úlohu prohledávání stavového prostoru (S, A') s počátečním stavem $\mathcal{I}$ a množinou koncových stavů G':

- Množina **stavů** $S = 2^P$ (potenční množina P)
  - stavy chápeme jako **množiny atomů**,
    - ★ de-facto výrokové proměnné, které v daném stavu platí,
- Množina **akčních instancí**
  $A' = \{(s, s') \in S \times S \mid \text{pre}(a) \subseteq s \wedge s' = (s \setminus \text{del}(a)) \cup \text{add}(a)\}$
  - **Akce** a je aplikovatelná ve stavu s právě tehdy, když $\text{pre}(a) \subseteq s$,
  - **Aplikace akce** a ve stavu s vede na stav $s' = (s \setminus \text{del}(a)) \cup \text{add}(a)$.
- **Počáteční stav** $\mathcal{I}$ přesně odpovídá počátečnímu stavu úlohy ve STRIPS,
- Množina **koncových stavů** $G' = \{s \in S | s \subseteq G\}$
  - obsahuje ty stavy, v nichž jsou splněny všechny cílové atomy

Plán ve STRIPS

Definice: Aplikace akce ve STRIPS

Uvažujme problém ve STRIPS $(P, A, \mathcal{I}, G)$. **Aplikace akce** $a \in A$ je zobrazení $\text{app}_a \colon 2^P \to 2^P$ dané jako

$$\text{app}_a(s) = (s \setminus \text{del}(a)) \cup \text{add}(a)$$

Definice: Plán ve STRIPS

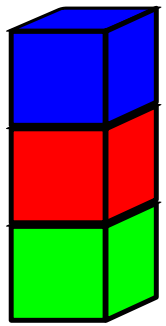
Uvažujme problém ve STRIPS $(P, A, \mathcal{I}, G)$. **Plán** pro $(P, A, \mathcal{I}, G)$ je posloupnost akcí $\pi = (a_1, \ldots, a_n), \forall i \in \{1, \ldots, n\} \colon a_i \in A$ taková, že

1. $\text{pre}(a_1) \subseteq \mathcal{I}$,
2. $\forall i \in \{2, \ldots, n\} \colon \text{pre}(a_i) \subseteq \text{app}_{a_{i-1}}(\text{app}_{a_{i-2}}(\ldots \text{app}_{a_1}(\mathcal{I}) \ldots))$,
3. $G \subseteq \text{app}_{a_n}(\text{app}_{a_{n-1}}(\ldots \text{app}_{a_1}(\mathcal{I}) \ldots))$

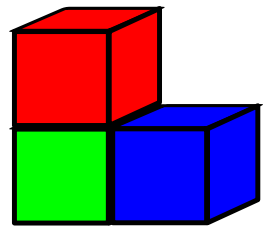
Plán o nejkratší možné délce nazveme **optimální** a budeme jej značit π^*.

Jazyk STRIPS: Příklad

```
  red-on-ground = 0
  red-on-top    = 0
  red-on-green  = 1
  red-on-blue   = 0
blue-on-ground  = 0
blue-on-top     = 1
blue-on-red     = 1
blue-on-green   = 0
green-on-ground = 1
green-on-top    = 0
green-on-red    = 0
green-on-blue   = 0
```



```
move-blue-red-ground
  pre = { blue-on-top, blue-on-red }
  add = { blue-on-ground,
          red-on-top }
  del = { blue-on-red }
```



```
  red-on-ground = 0
  red-on-top    = 1
  red-on-green  = 1
  red-on-blue   = 0
blue-on-ground  = 1
blue-on-top     = 1
blue-on-red     = 1
blue-on-green   = 0
green-on-ground = 1
green-on-top    = 0
green-on-red    = 0
green-on-blue   = 0
```

```
move-blue-ground-red
  pre = { blue-on-ground,
          blue-on-top,
          red-on-top }
  add = { blue-on-red }
  del = { blue-on-ground,
          red-on-top }
```

Jazyk STRIPS: Příklad

$$P = \{\textit{red-on-ground}, \textit{red-on-top}, \textit{red-on-green}, \textit{red-on-blue},$$
$$\textit{green-on-ground}, \textit{green-on-top}, \textit{green-on-red}, \textit{green-on-blue},$$
$$\textit{blue-on-ground}, \textit{blue-on-top}, \textit{blue-on-red}, \textit{blue-on-green}\}$$

$$A = \{\textit{move-red-ground-green}, \textit{move-red-ground-blue}, \textit{move-red-green-blue},$$
$$\textit{move-red-blue-green}, \textit{move-red-green-ground}, \textit{move-red-blue-ground},$$
$$\textit{move-green-ground-red}, \textit{move-green-ground-blue}, \ldots\}$$

$$\mathcal{I} = \{\textit{red-on-green}, \textit{green-on-ground}, \textit{blue-on-top}, \textit{blue-on-red}\}$$

$$G = \{\textit{green-on-red}, \textit{red-on-ground}, \textit{blue-on-top}, \textit{blue-on-green}\}$$

$$\pi^* = (\textit{move-blue-red-ground}, \textit{move-red-green-ground}, \textit{move-green-ground-red},$$
$$\textit{move-blue-ground-green})$$

STRIPS a predikátová logika

- STRIPS pracuje na úrovni výrokové logiky – atomy buďto platí, anebo neplatí,
- jazyk je poměrně jednoduchý a přitom dostatečně expresivní pro popis plánovacích problémů,
  - řešení problému v jazyce STRIPS není jednodušší, než řešení obecného plánovacího problému,
- struktura problému ve STRIPS umožňuje efektivní **návrh obecných plánovacích algoritmů**
  - práce s množinou atomů dává algoritmu možnost odhalit strukturu problému, byť je problém obecný – uvidíme dále,
- definice problému na úrovni atomů a konkrétních akcí je však nepraktická
  - STRIPS problémy proto v praxi definujeme pomocí **predikátové logiky**

STRIPS a predikátová logika

- **Predikátová logika** = jeden ze základních kamenů racionálně uvažujících a jednajících systémů v AI
  - Umožňuje popsat a modelovat obrovskou řadu domén
    - ★ uplatnění všude: od filosofie přes lingvistiku až po computer science
  - BI-MLO je předmět s mnohem větším přesahem, než by se mohlo zdát :-)
- Z hlediska plánování nás zejména zajímají následující koncepty predikátové logiky:
  - **univerzum** objektů, v jehož kontextu je jsou interpretovány klauzule,
  - **predikáty**, tedy relace na univerzu,
  - **substituce**, tedy náhrada všech výskytů jedné proměnné jinou proměnnou,
- S využitích těchto konceptů jsme schopni formulovat problémy v jazyce STRIPS kompaktnější formou

STRIPS a predikátová logika: Příklad

Zavedeme:

- **Univerzum** objektů: $U = \{R, G, B\}$,
- **Predikáty**: $P = \{on, on\text{-}ground, on\text{-}top, distinct\}$,

Následně zformulujeme problém ve STRIPS $(P', A, \mathcal{I}, G)$ jako:

$$P' = \{on(x, y), on\text{-}ground(x), on\text{-}top(x), distinct(x, y) \mid x, y \in U\},$$

$$A = \{move(what, from, to), from\text{-}ground(what, to), to\text{-}ground(what, from)\}$$

$$\mathcal{I} = \{on\text{-}ground(G), on(G, R), on(R, B), on\text{-}top(B)\} \cup \{distinct(x, y) \mid x, y \in U, x \neq y\}$$

$$G = \{on\text{-}ground(R), on(R, G), on(G, B), on\text{-}top(B)\},$$

STRIPS a predikátová logika: Příklad

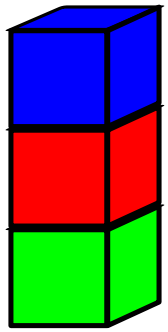
Pro jednotlivé akce pak definujeme předpoklady, „add“ efekty a „delete“ efekty:

- *move*(*what*, *from*, *to*)
  - pre(*move*) = {*on*(*from*, *what*), *on-top*(*what*), *on-top*(*to*), *distinct*(*what*, *to*)},
  - add(*move*) = {*on*(*to*, *what*), *on-top*(*from*)},
  - del(*move*) = {*on*(*from*, *what*), *on-top*(*to*)},
- *from-ground*(*what*, *to*)
  - pre(*from-ground*) = {*on-ground*(*what*), *on-top*(*what*), *on-top*(*to*)}
  - add(*from-ground*) = {*on*(*to*, *what*)}
  - del(*from-ground*) = {*on-ground*(*what*), *on-top*(*to*)}
- *to-ground*(*what*, *from*)
  - pre(*to-ground*) = {*on*(*from*, *what*), *on-top*(*what*)}
  - add(*to-ground*) = {*on-ground*(*what*), *on-top*(*from*)}
  - del(*to-ground*) = {*on*(*from*, *what*)}

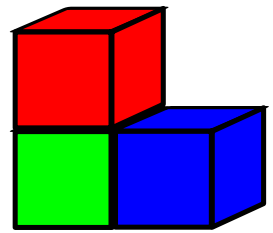
STRIPS a predikátová logika: Příklad

```
on-ground(green)
on(green,red)
on(red,blue)
on-top(blue)

distinct(red,green)
distinct(green,red)
...
```



```
to-ground(what,from)
[what:=blue,from:=red]
  pre = { on(from,what),
          on-top(what) }
  add = { on-ground(what),
          on-top(from) }
  del = { on(from,what) }
```



```
on-ground(green)
on(green,red)
on-top(blue)
on-ground(blue)
on-top(red)

distinct(red,green)
distinct(green,red)
...
```

```
from-ground(what,to)
[what:=blue,to:=red]
  pre = { on-ground(what),
          on-top(what),
          on-top(to) }
  add = { on(to,what) }
  del = { on-ground(what),
          on-top(to) }
```

Řešení problémů ve STRIPS

- Viděli jsme, že STRIPS je poměrně mocný jazyk pro definici plánovacích problémů,
- Připomeňme:

$$\text{Problém} \longrightarrow \text{Model} \longrightarrow \text{Jazyk} \longrightarrow \text{Solver} \longrightarrow \text{Řešení},$$

- v našem případě

$$\text{Problém} \longrightarrow \text{Klasické plánování} \longrightarrow \text{STRIPS} \longrightarrow \underbrace{\text{Solver}}_{?} \longrightarrow \text{Řešení}$$

- Různé jazyky pro zápis problémů mají svá specifika a vedou na tvorbu **jazykově-specifických algoritmů**
- Dále se budeme zabývat pouze algoritmy pro jazyk STRIPS

Typy plánovacích algoritmů

Plánovací (i obecně prohledávací) algoritmy lze charakterizovat podle toho, jak prohledávají stavový prostor:

- **optimální** vs. **vyhovující**
  - **optimální** – požadujeme nejkratší možný plán,
  - **vyhovující** (angl. *satisfactory*) – požadujeme jakýkoli plán, ne nutně optimální (např. cesta k chybě),
- **dopředné** vs. **zpětné** vs. **obousměrné**
  - **dopředné** (progresivní) – hledáme cestu z počátečního do nějakého z cílových stavů
    - ★ přístup, kterým se zabýváme v BI-ZUM, avšak nikoli jediný možný!
  - **zpětné** (regresní) – hledáme cesty z množiny cílových stavů do stavu počátečního
  - **obousměrné** (angl. *bidirectional*) – kombinace obou přístupů, hledáme stav, kde se prohledávací stromy protnou
- **neinformované** vs. **heuristické**

Nejčastěji používáme heuristické dopředné prohledávání. Problém představuje návrh heuristik.

Heuristika STRIPS (Fikes & Nilsson, 1971)

- Jednoduchá heuristika, která uvažuje počet cílových atomů $G = \{g_1, \ldots, g_\ell\}$, které nejsou ve zkoumaném stavu splněny:

$$h(s) = |G \setminus s|$$

- Intuitině správná: čím méně cílových atomů zbývá splnit, tím blíže jsme koncovému stavu
- Má však celou řadu nedostatků:
  - **nízká informativnost** – pro většinu následníků vrací stejný heuristický odhad,
  - **citlivost na formulaci** problému – v extrémním případě

$$(P, A, \mathcal{I}, G) \longmapsto (P \cup \{x\}, A \cup \{a_x\}, \mathcal{I}, \{x\}),$$

    kde $\text{pre}(a_x) = G$, $\text{add}(a_x) = \{x\}$, $\text{del}(a_x) = \{\}$,
  - **ignoruje strukturu** problému – nebere v potaz množinu akcí

Relaxace a abstrakce

Obecný přístup k tvorbě heuristik: **Cena řešení zjednodušené verze problému.**

Dva základní soudobé přístupy k tvorbě heustik pro problémy ve STRIPS jsou:

- **relaxace**
  - uvažuje problém stejné velikosti s méně omezujícími podmínkami,
  - používá se zejména pro **vyhovující** plánování,
  - heuristiky vesměs nejsou přípustné,
- **abstrakce**
  - uvažuje menší problém se stejnými omezeními,
  - využití pro **optimální** plánování,
  - heuristiky jsou zásadně přípustné,

Relaxace

- Úkol: odhadnout cenu plánu ze stavu h do nějakého z koncových stavů
  - o problému nemáme a priori žádné znalosti kromě jeho formulace ve STRIPS,
- Idea řešení – rozdělme efekty akcí na **„dobré“** a **„špatné“**,
  - „dobré“ efekty nám pomáhají směřovat k cíli,
  - „špatné“ efekty jsou nežádoucí vedlejší účinky akcí,
- Příklad pro problém „Lišák“, pohyb kamene z pole x na pole y:
  - „dobrý“ efekt: pole x je nyní volné,
  - „špatný“ efekt: pole y již není volné,
- Výhoda STRIPS: snadno odlišíme dobré efekty od špatných:
  - **add** efekty jsou **„dobré“** – přidávají nově platné atomy,
  - **del** efekty jsou **„špatné“** – odebírají platné atomy,
  - to proto, že všechny předpoklady $\text{pre}(a_i)$ jsou v **pozitivní formě**
    - ★ vždy vyžadujeme pouze platnost atomů, nikdy jejich neplatnost

Relaxace: Definice

Definice: Relaxace akce

Uvažujme STRIPS akci danou jako $a = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$. **Relaxace akce** a je akce $a^+ = \langle \text{pre}(a), \text{add}(a), \{\} \rangle$.

Definice: Relaxace plánovací úlohy

Uvažujme plánovací úlohu ve STRIPS $\Pi = (P, A, \mathcal{I}, G)$. **Relaxace plánovací úlohy** Π je plánovací úloha $\Pi^+ = (P, \{a^+ | a \in A\}, \mathcal{I}, G)$.

Definice: Relaxace plánu

Uvažujme plán ve STRIPS π. **Relaxací plánu** $\pi = (a_1, a_2, \ldots, a_n)$ je plán $\pi^+ = (a_1^+, a_2^+, \ldots, a_n^+)$.

Pozorování: Pokud je π platným plánem pro Π, pak π^+ je platným plánem pro Π^+.

Hladový algoritmus pro relaxovanou plánovací úlohu

Algorithm 1 Greedy algorithm for $(P, A^+, \mathcal{I}, G)$

1: $s \leftarrow \mathcal{I}$
2: $\pi^+ \leftarrow \epsilon$
3: **while** $G \not\subseteq s$ **do**
4: **if** $\exists a \in A^+ : \text{pre}(a) \subseteq s \wedge \text{add}(a) \not\subseteq s$ **then**
5: append(π^+, a)
6: $s \leftarrow s \cup \text{add}(a)$
7: **else**
8: **return** unsolvable
9: **end if**
10: **end while**
11: **return** π^+

Heuristika h^+ a hladový algoritmus

Definice: Heuristika h^+

Uvažujme plánovací problém $\Pi = (P, A, I, G)$. Relaxovaná heuristika $h^+(s) : 2^P \rightarrow \mathbb{R}_0^+$ pro Π je dána jako:

$$h^+(s) = \arg \min_{(a_1^+,\ldots,a_n^+) \text{ je plán pro } \Pi^+} (n).$$

Tvrzení

Heuristika h^+ pro Π je přípustná.

Tvrzení

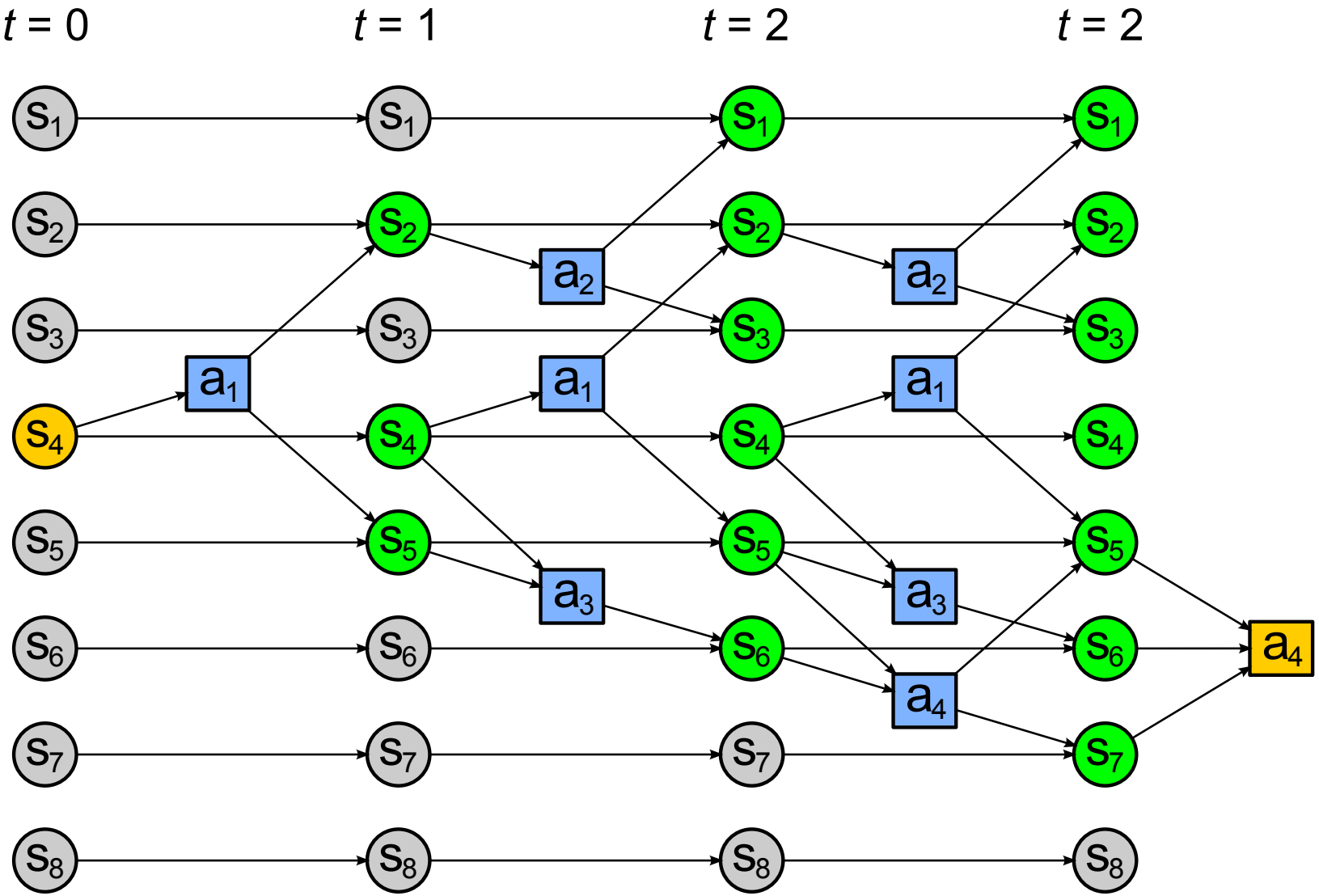
Hladový algoritmus **neumožňuje spočítat** h^+. Výpočet h^+ stále představuje NP-úplný problém (není znám deterministický algoritmus, který by jej v obecném případě vyřešil v polynomiálním čase).

Relaxační heuristiky v praxi

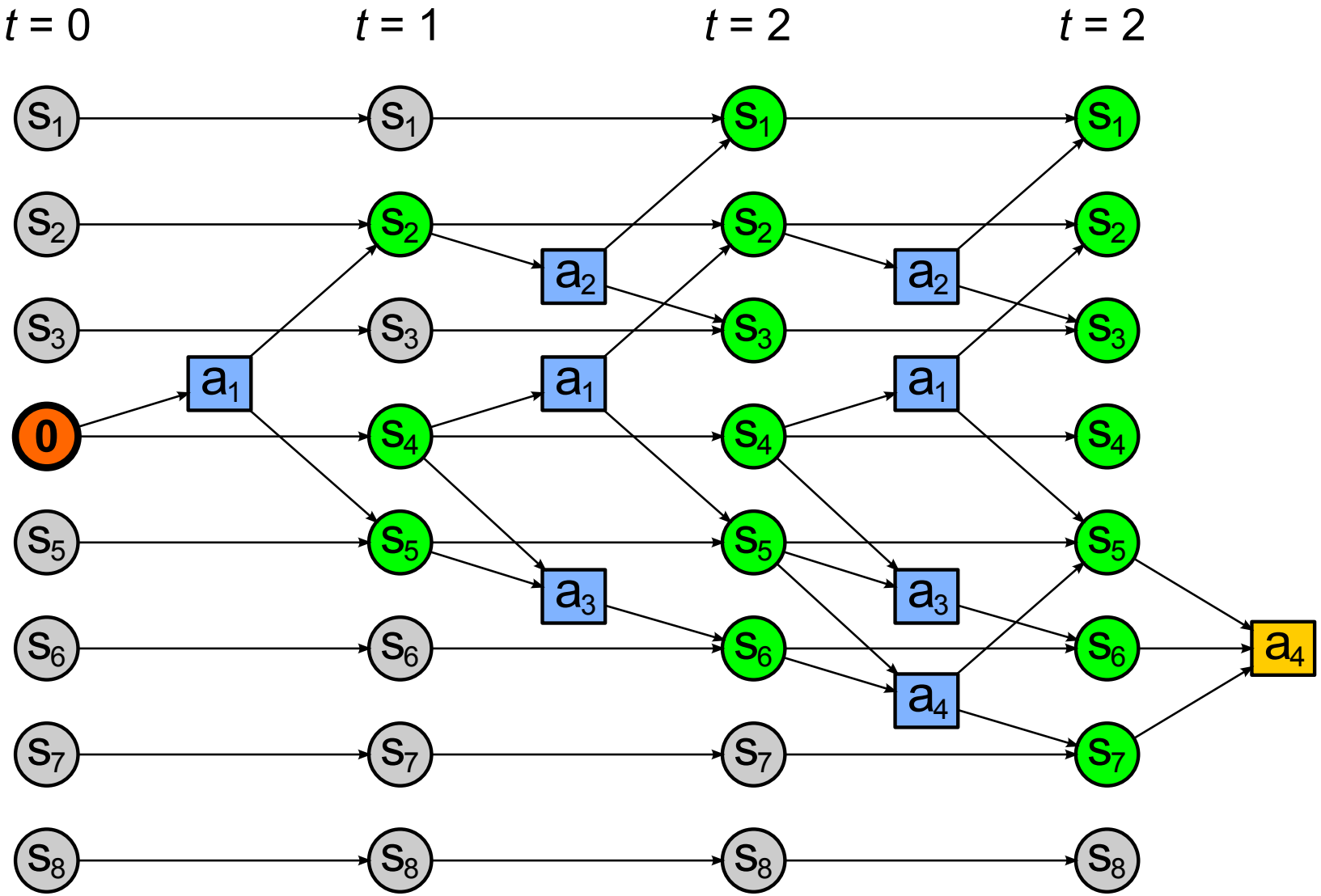
Jak jinak použít relaxovaný problém Π^+ k vytvoření heuristiky pro Π?

- Implementace **optimálního** plánovacího algoritmu pro Π^+, ačkoli je problém NP-úplný
  - heuristika $\mathbf{h}^+$, nerealistický přístup,
- Odhad ceny optimálního plánu pro Π^+ jiným způbem, než vyřešením Π^+
  - heuristika $\mathbf{h_{max}}$
  - heuristika $\mathbf{h_{add}}$
- Nalezení plánu pro Π^+, který není nezbytně optimální, ale je „rozumně“ suboptimální
  - heuristika $\mathbf{h_{FF}}$

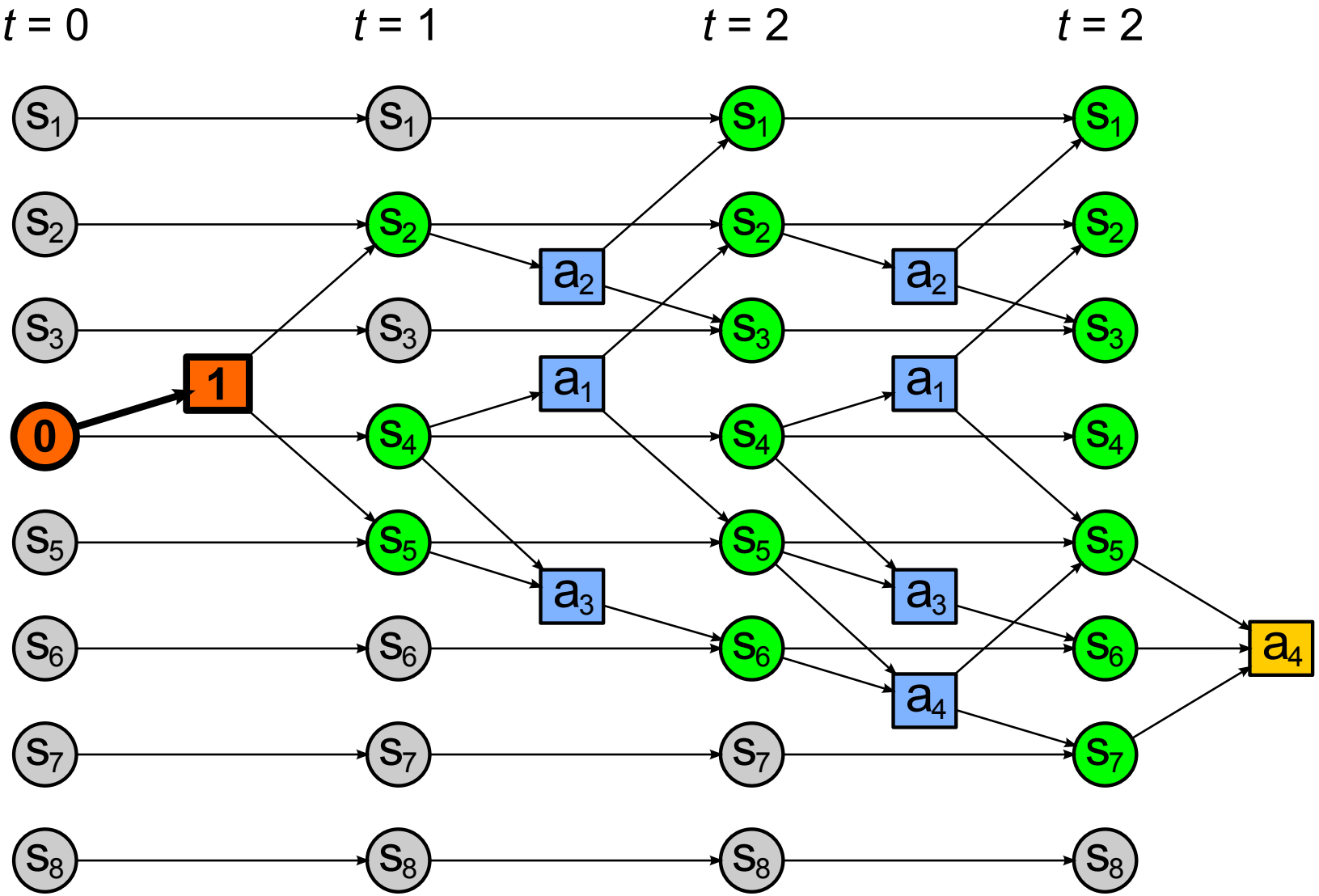
Heuristika h_{max}: Příklad



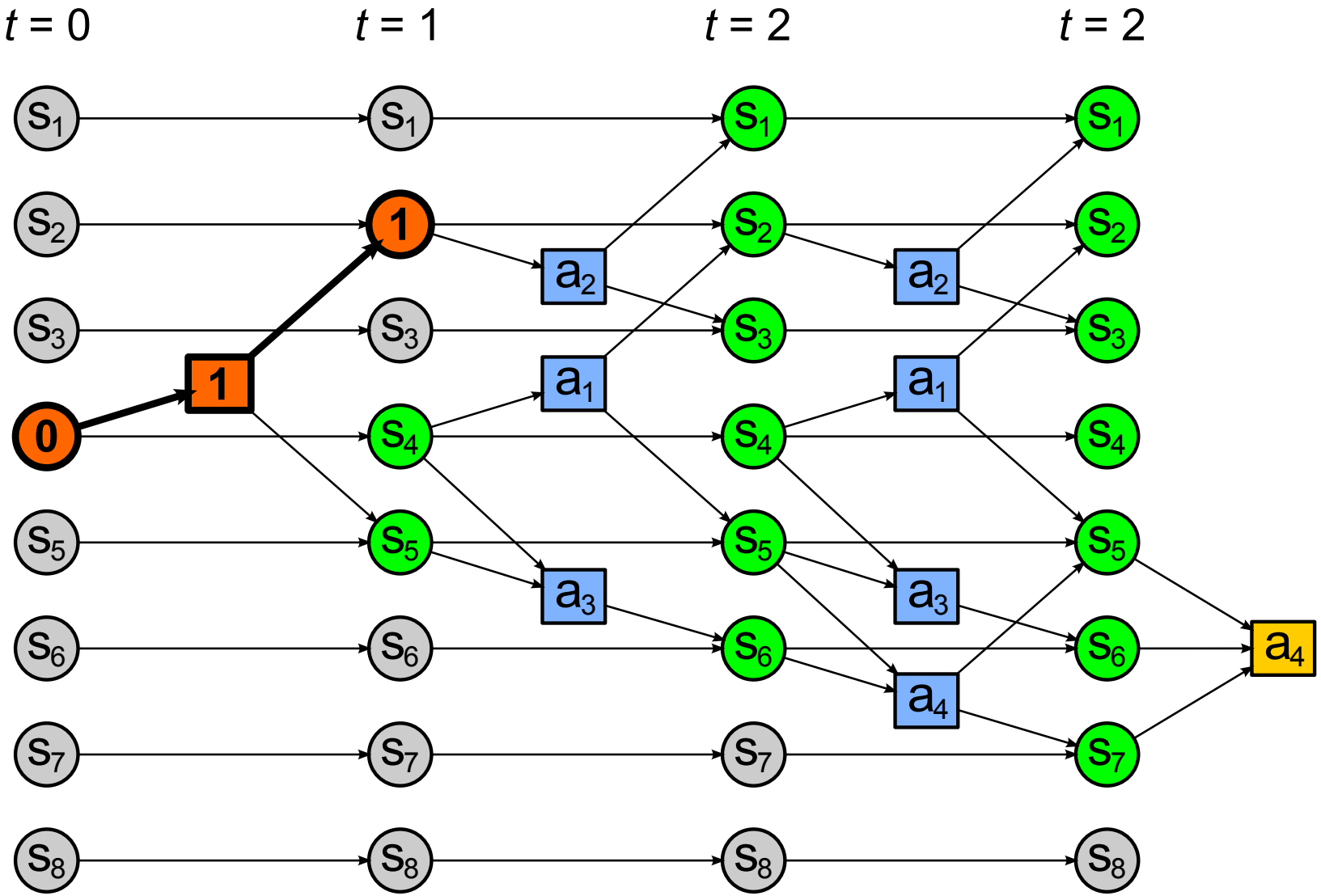
Heuristika h_{max}: Příklad



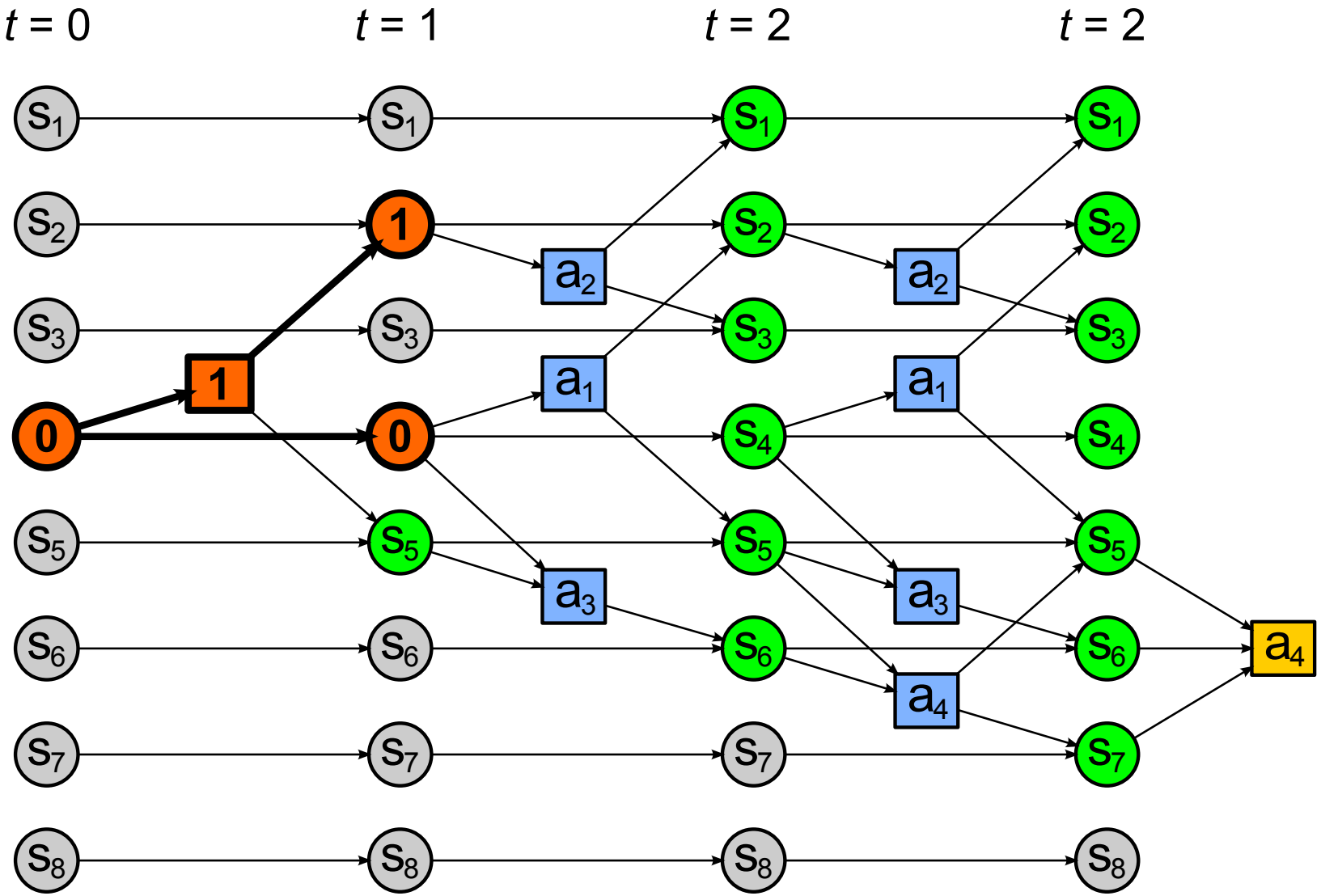
Heuristika h_{max}: Příklad



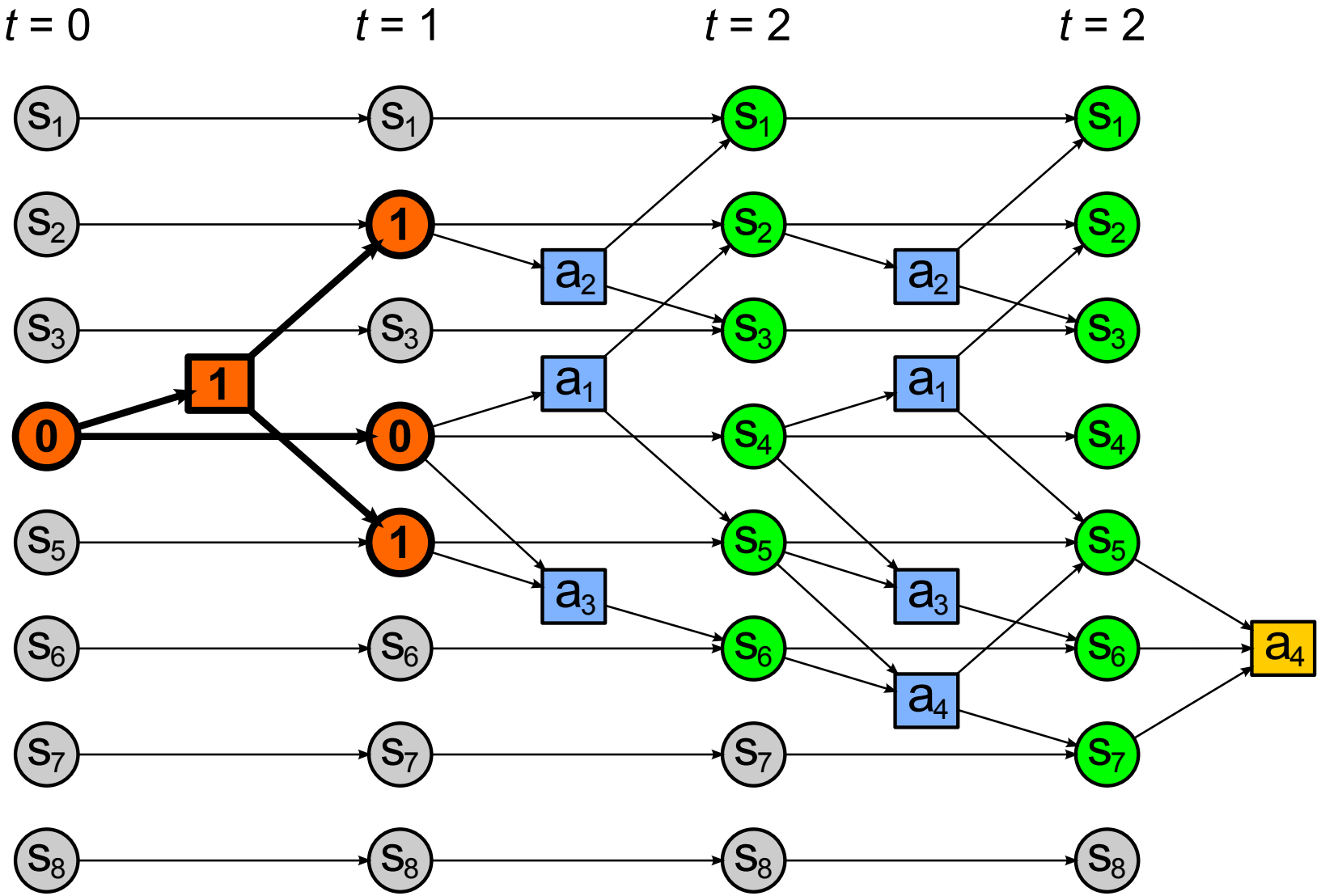
Heuristika h_{max}: Příklad



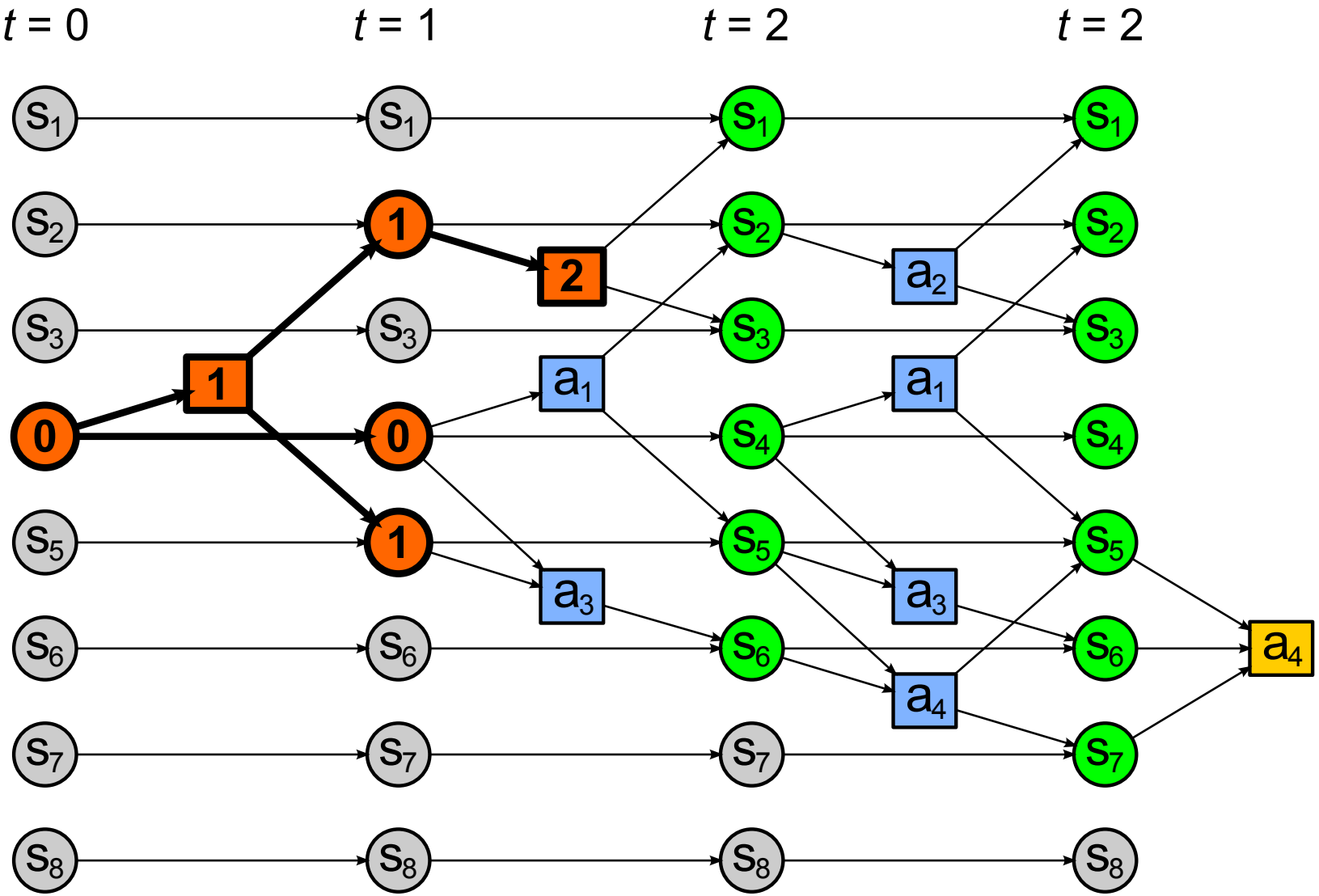
Heuristika h_{max}: Příklad



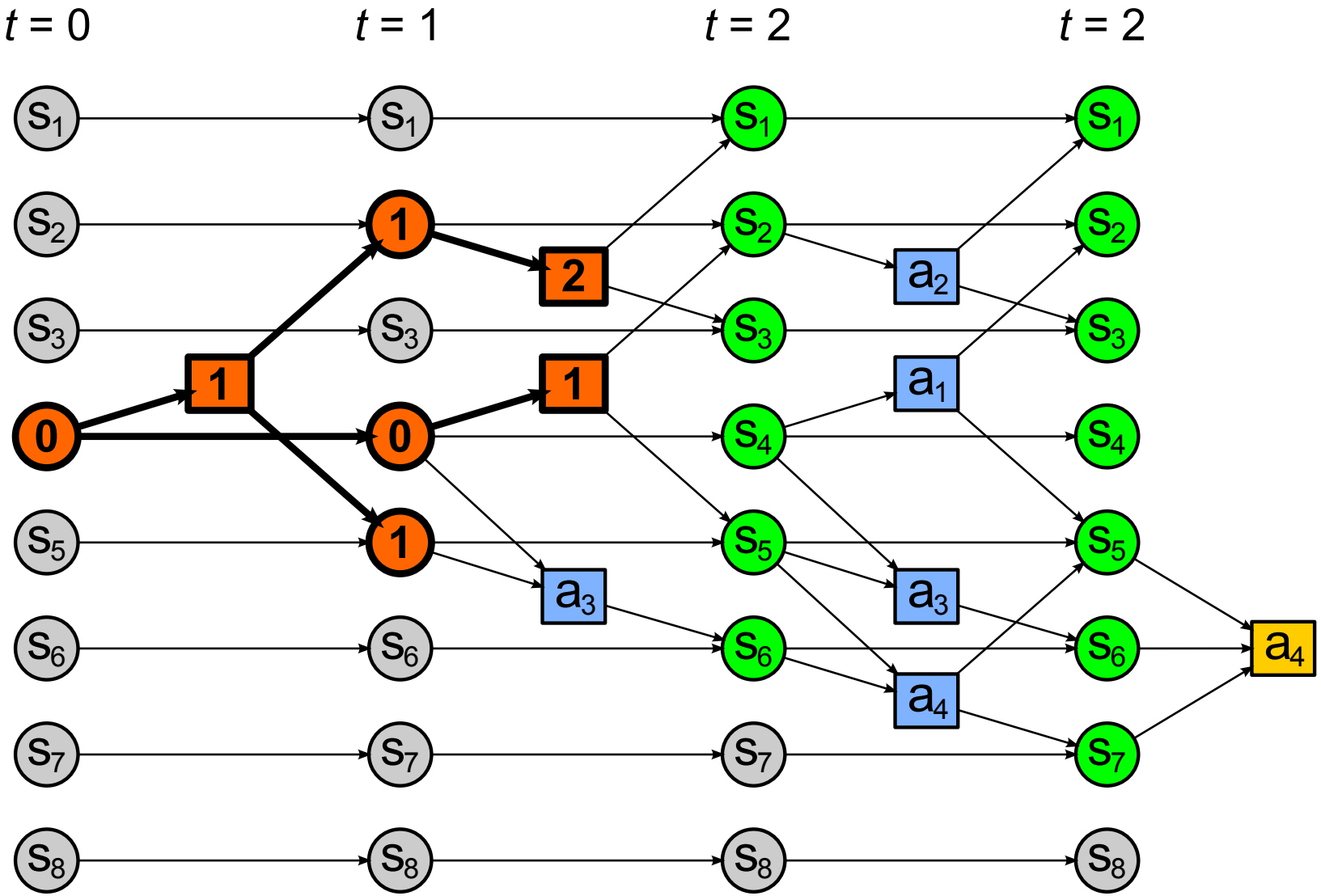
Heuristika h_{max}: Příklad



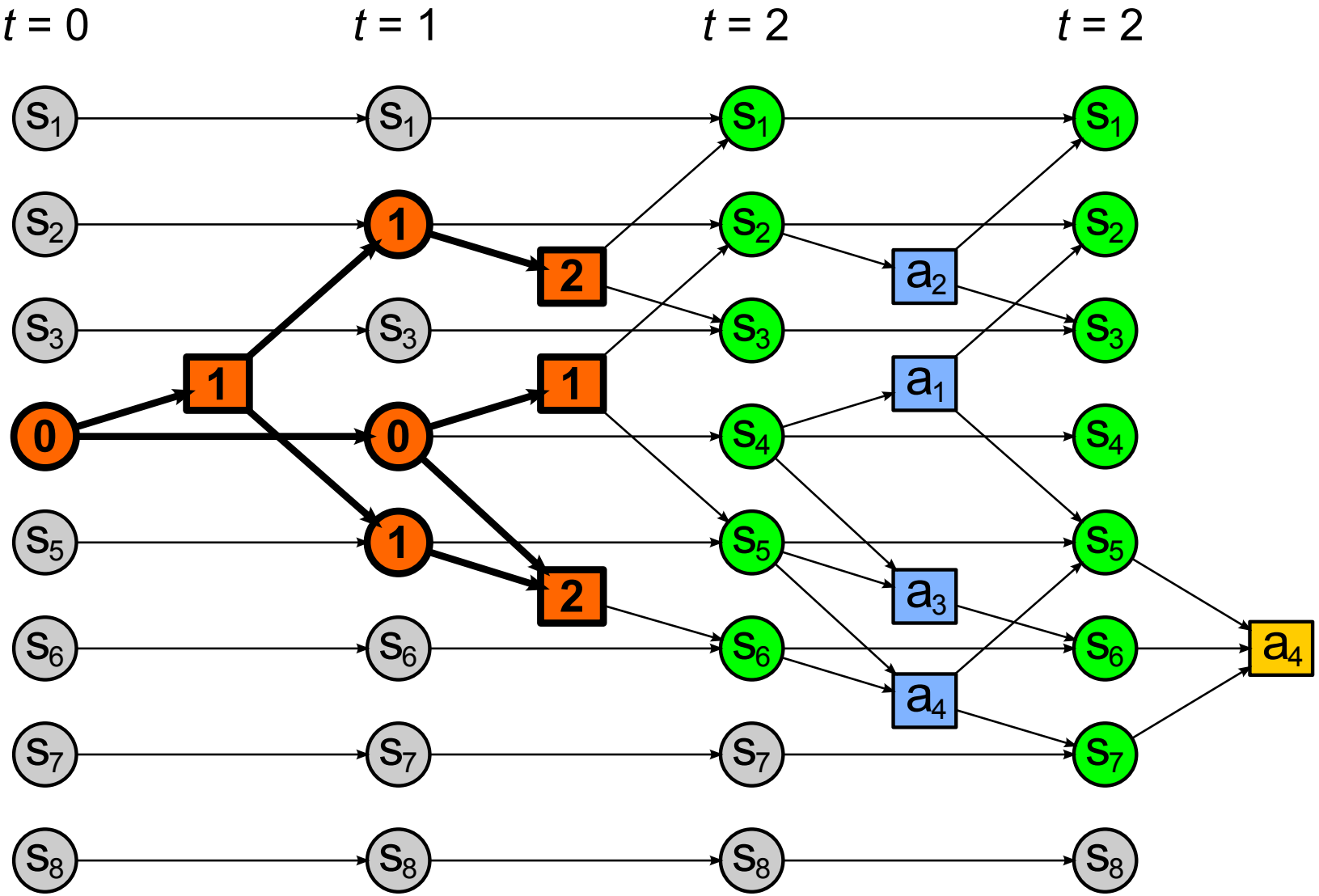
Heuristika h_{max}: Příklad



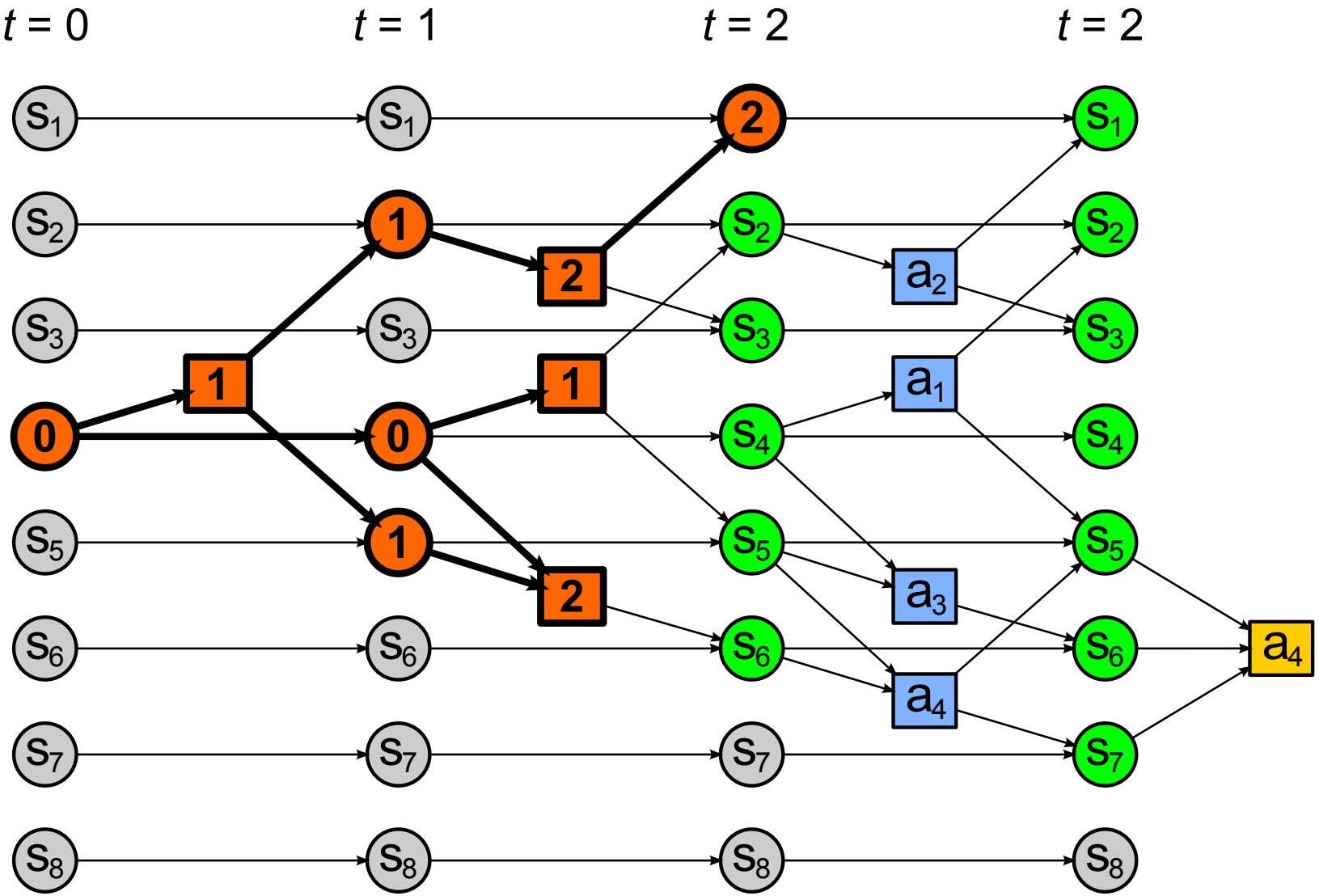
Heuristika h_{max}: Příklad



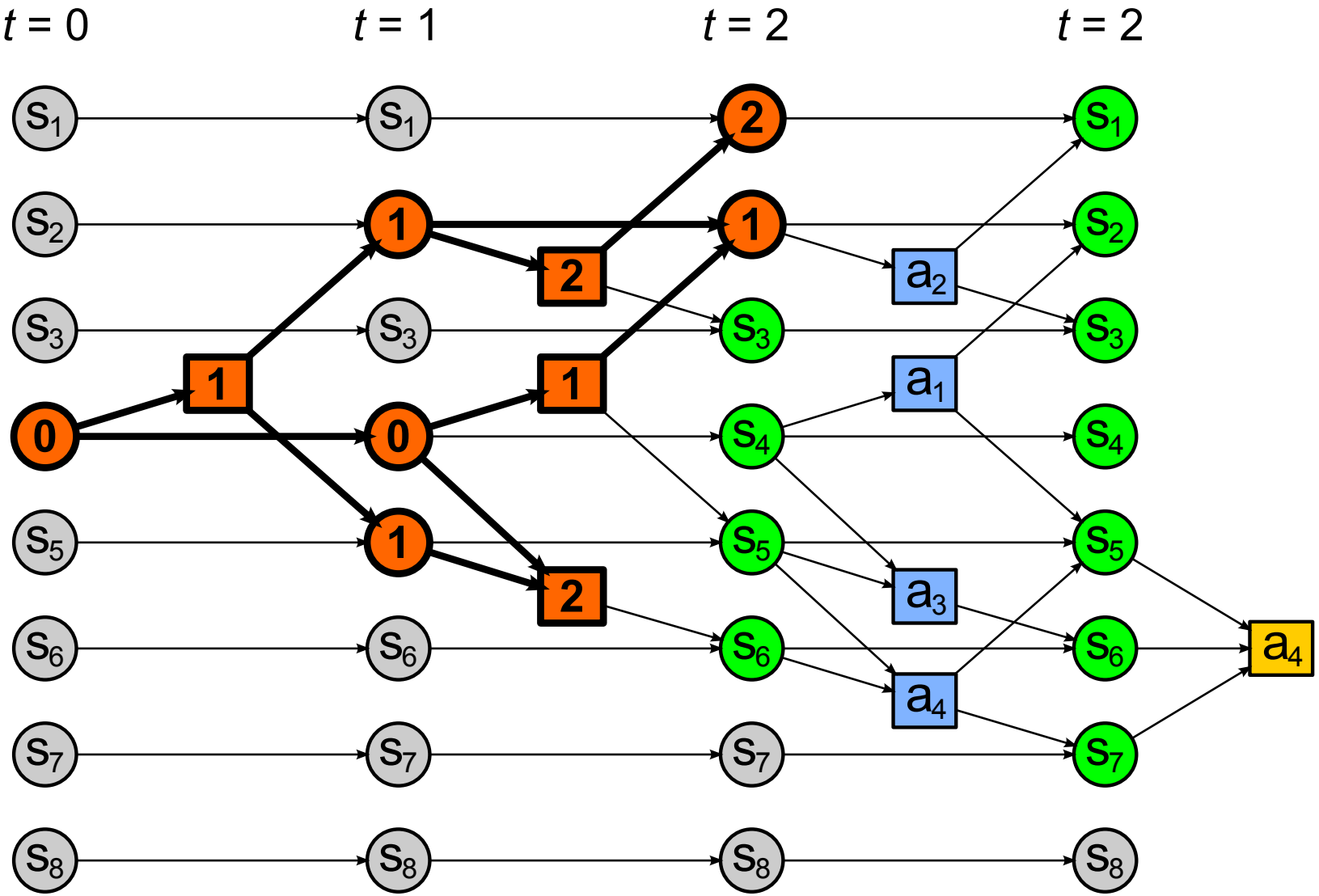
Heuristika h_{max}: Příklad



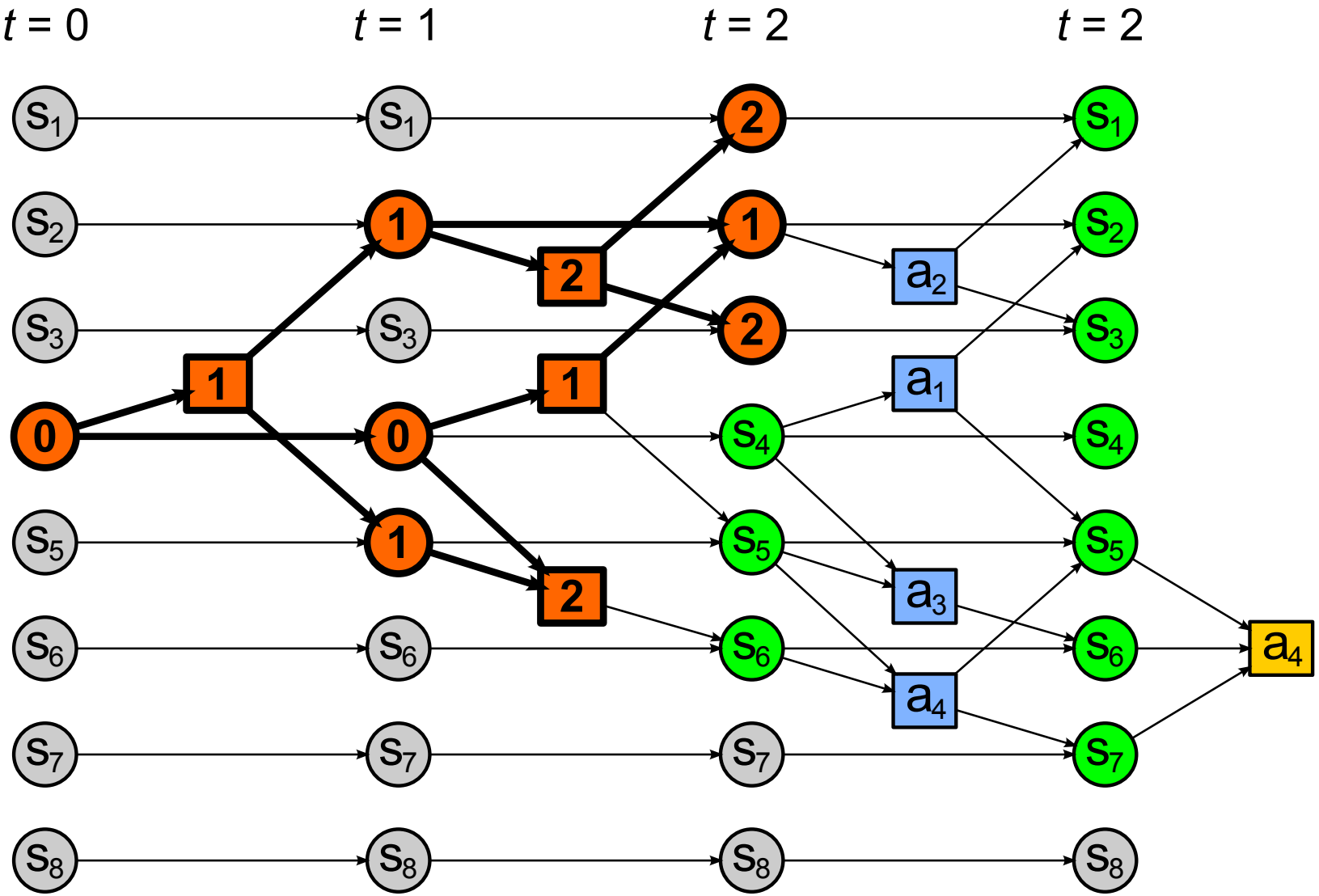
Heuristika h_{max}: Příklad



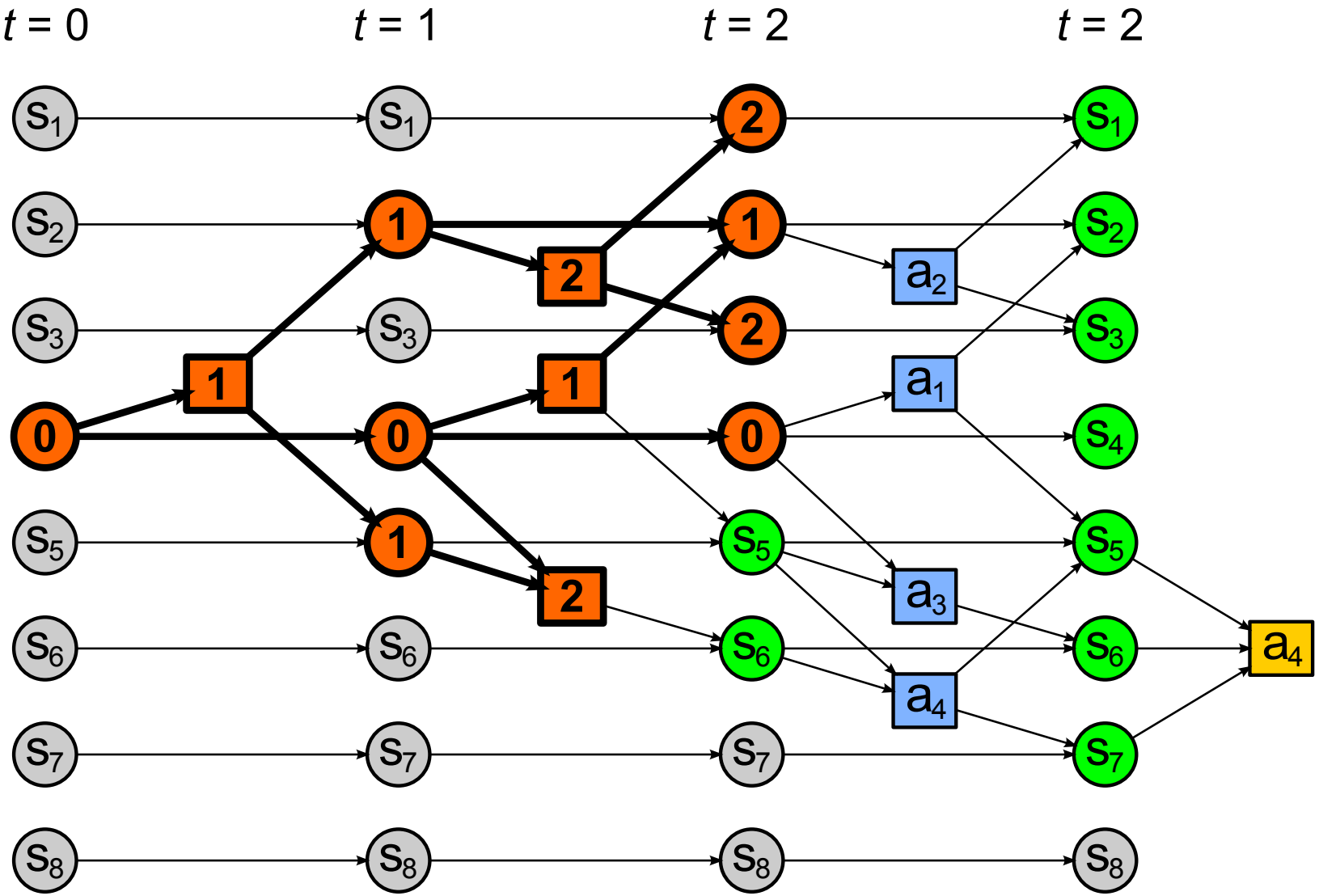
Heuristika h_{max}: Příklad



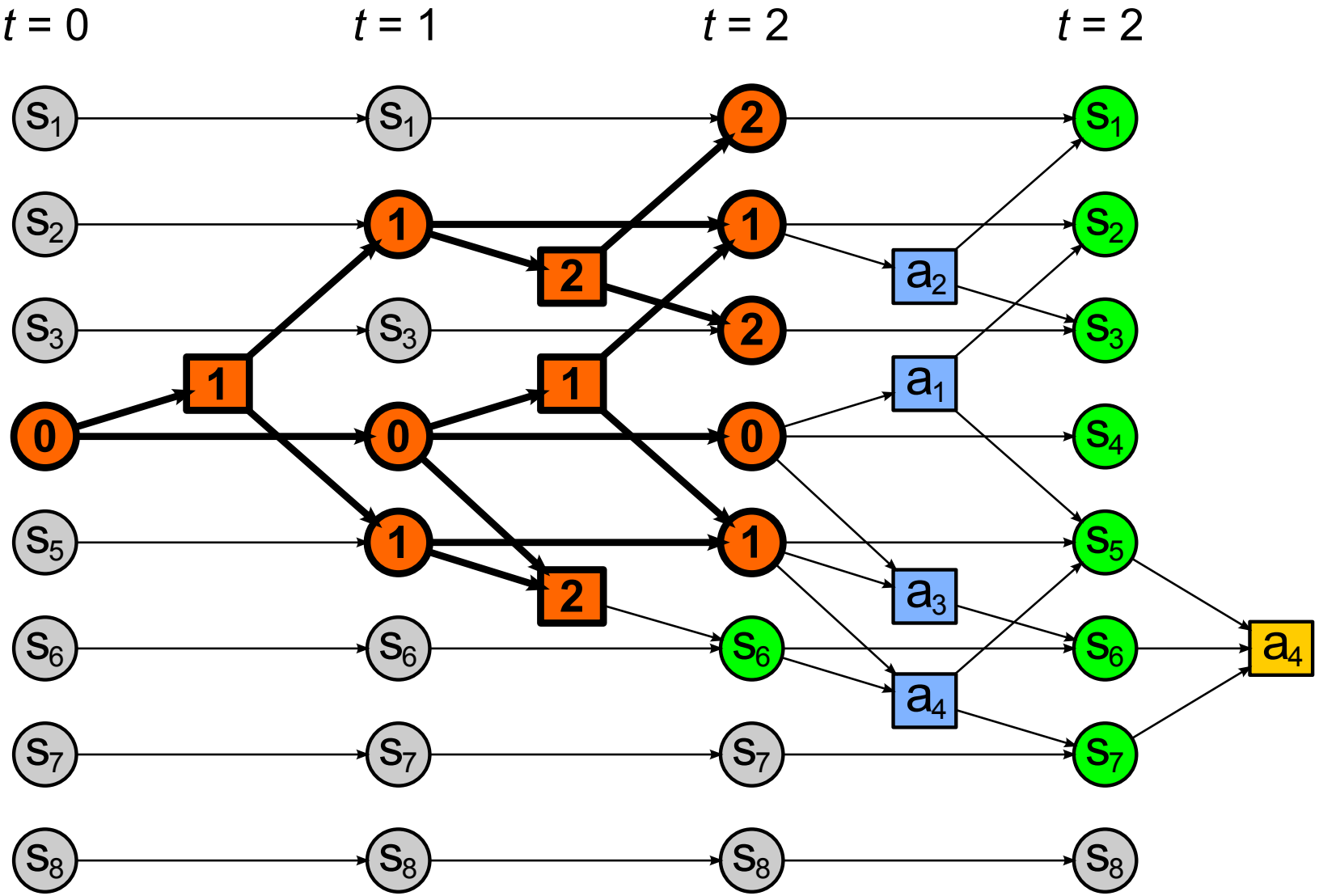
Heuristika h_{max}: Příklad



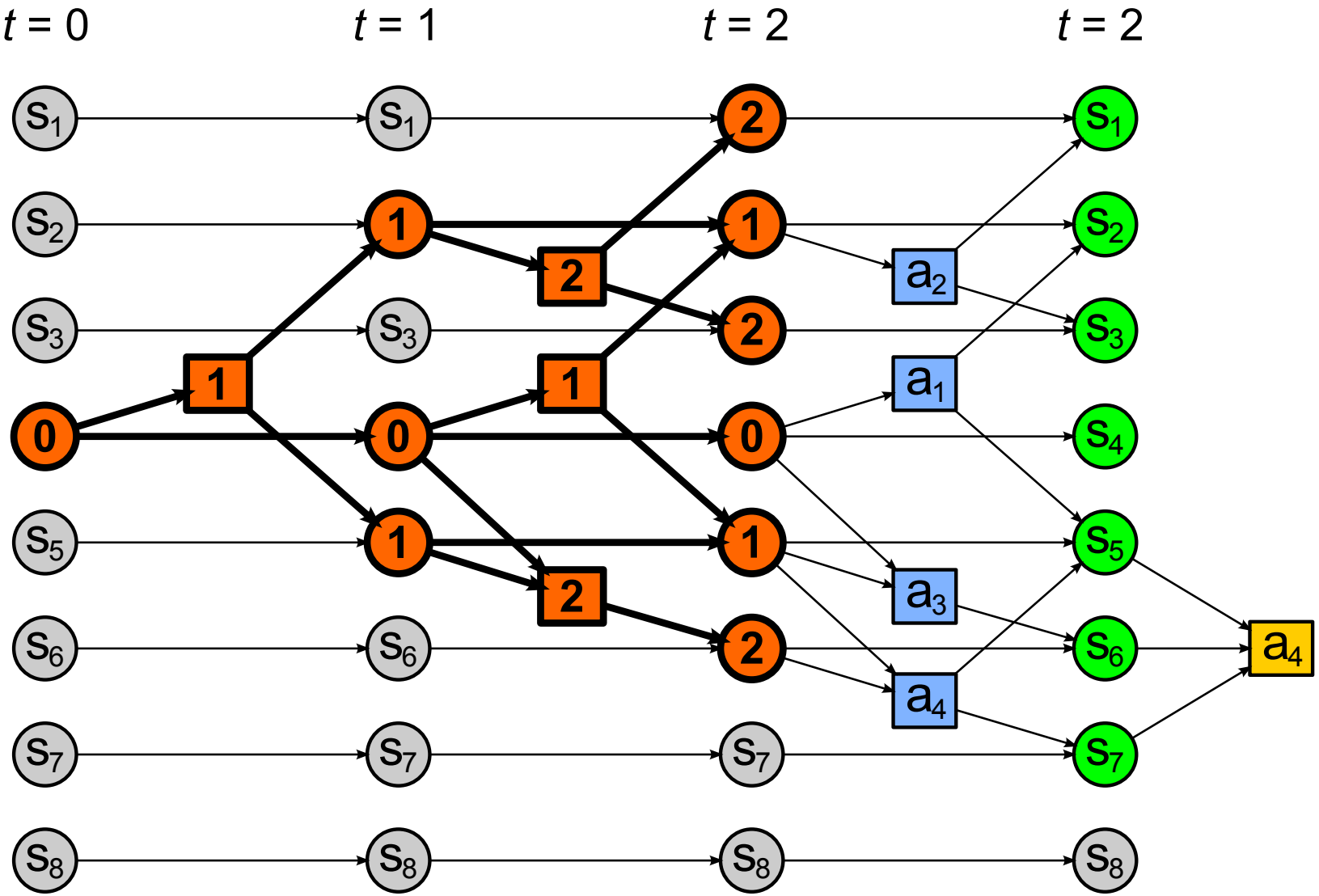
Heuristika h_{max}: Příklad



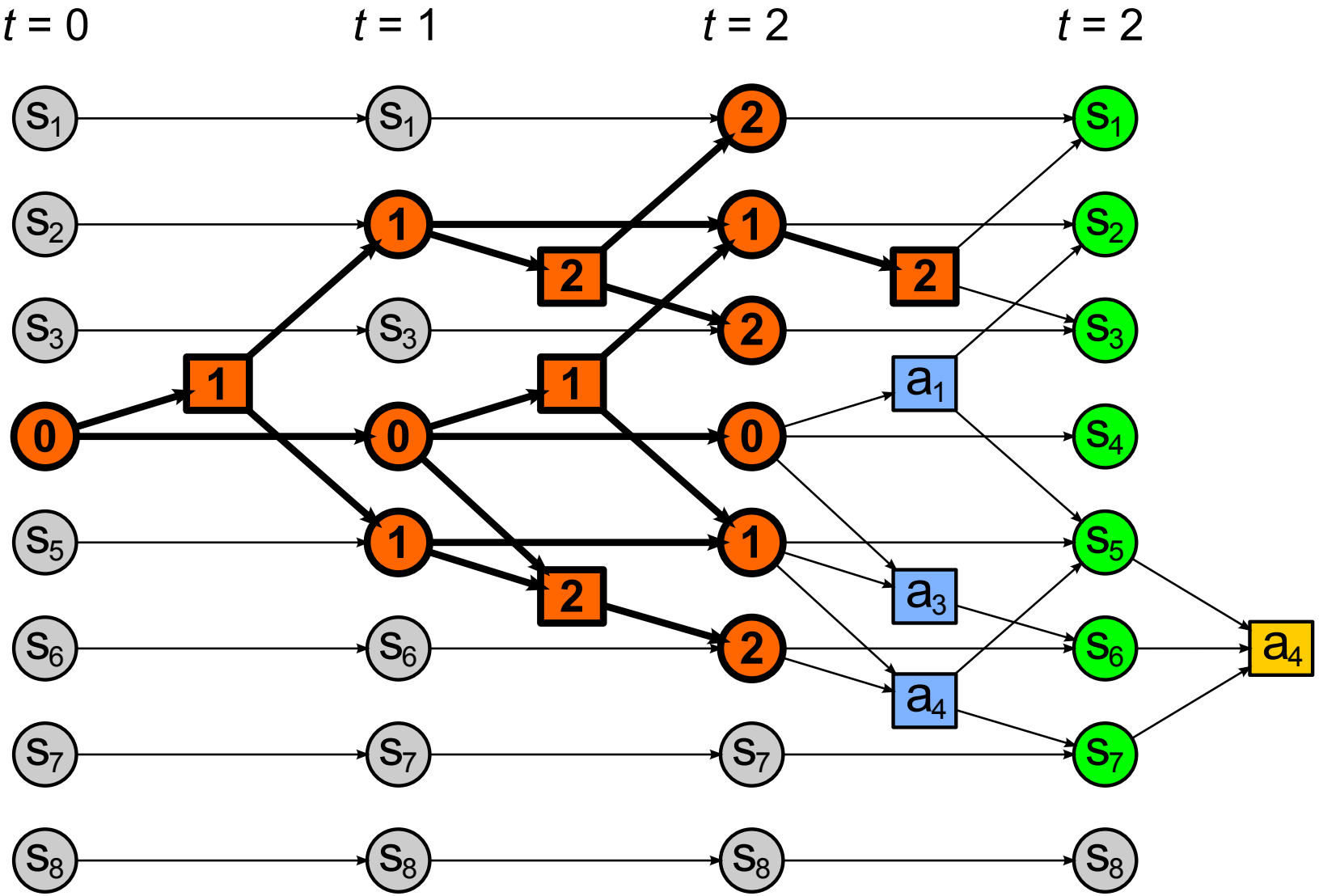
Heuristika h_{max}: Příklad



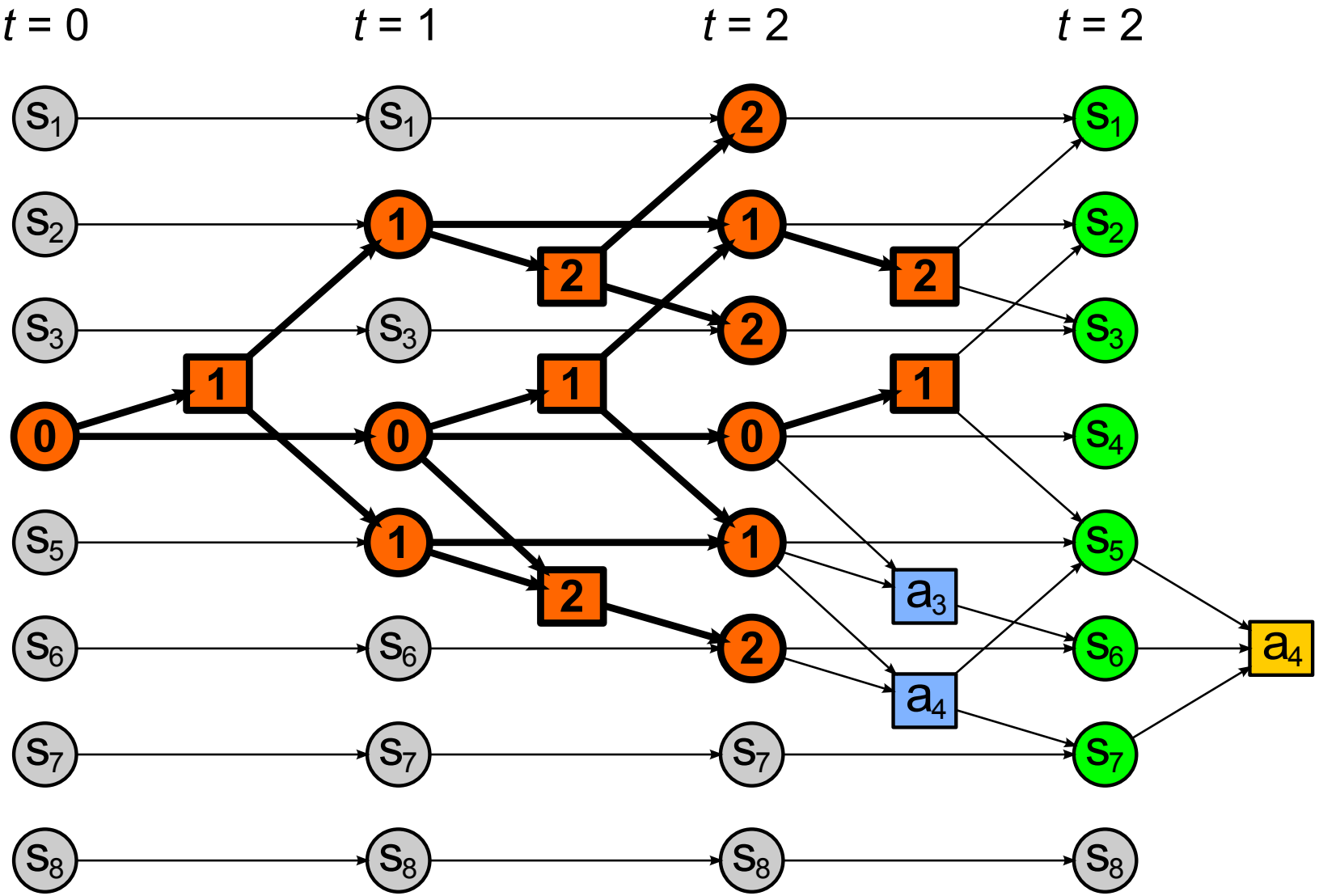
Heuristika h_{max}: Příklad



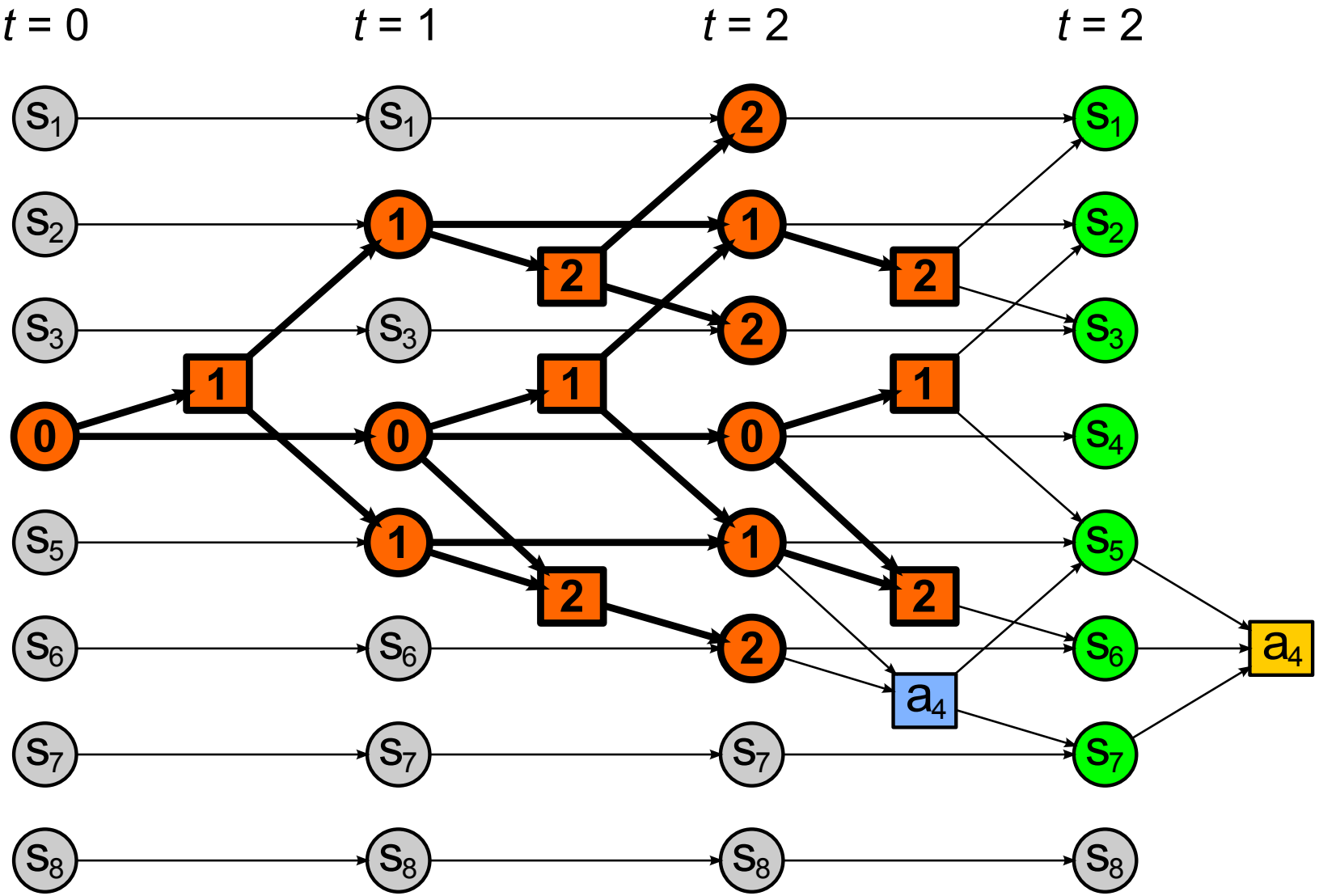
Heuristika h_{max}: Příklad



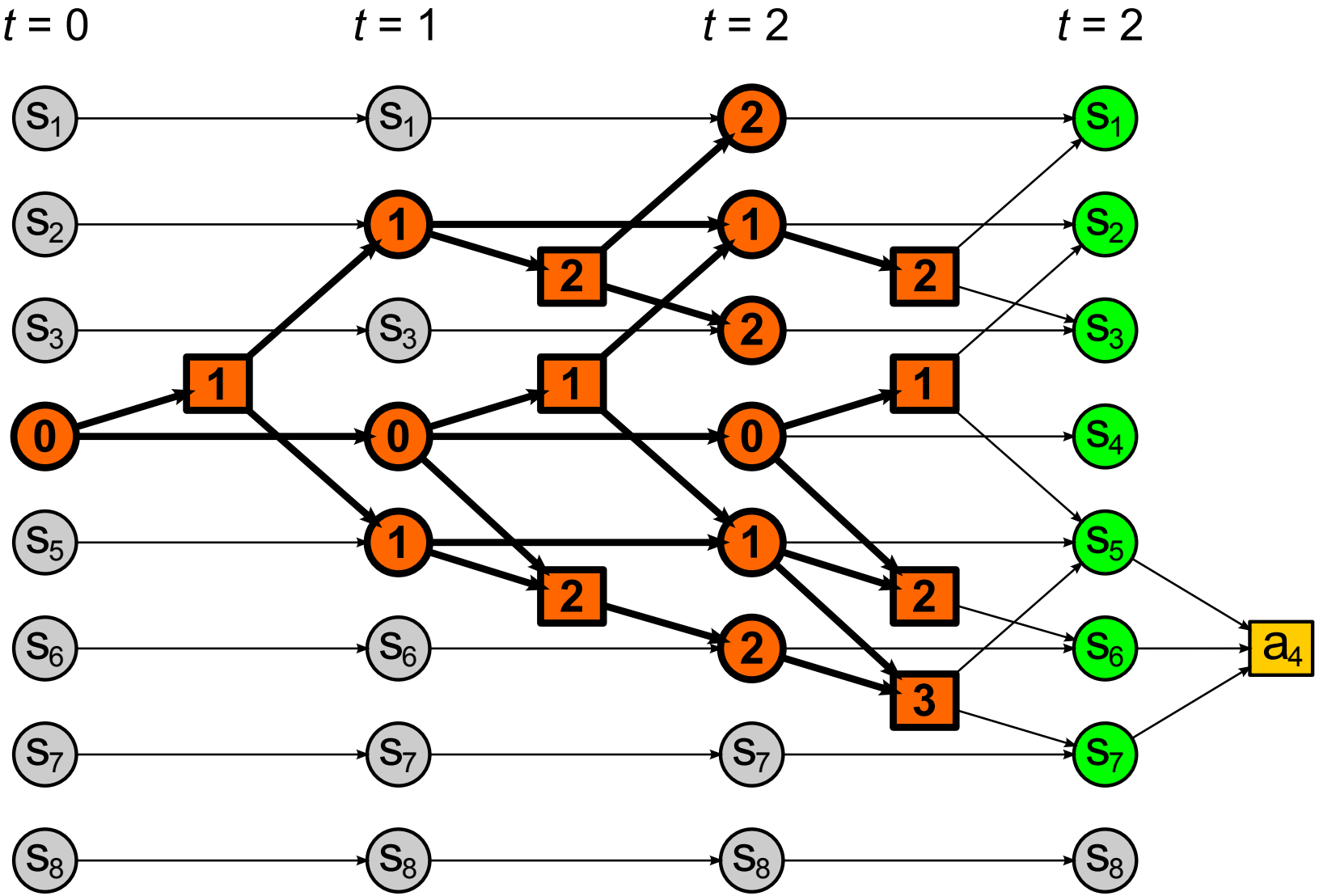
Heuristika h_{max}: Příklad



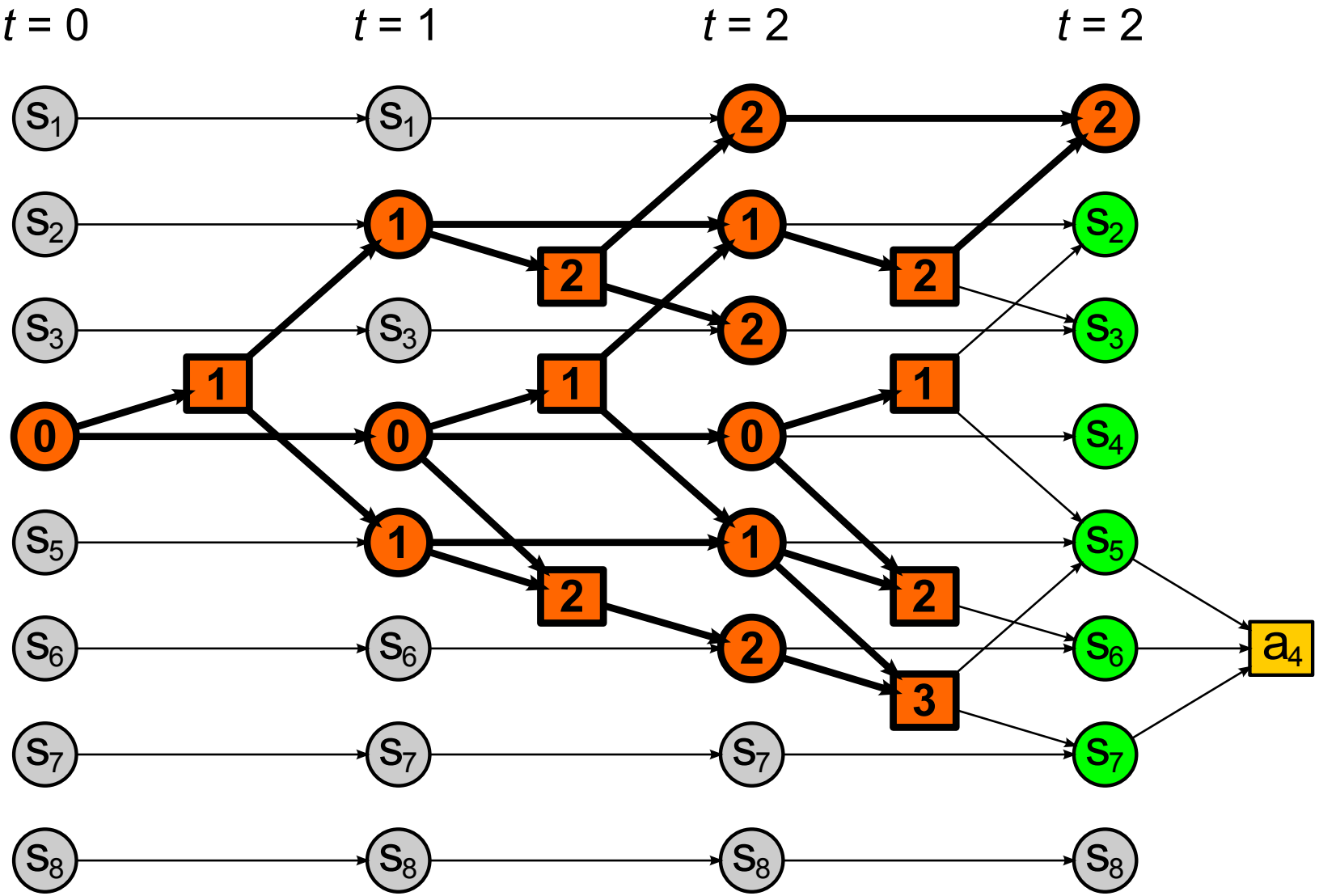
Heuristika h_{max}: Příklad



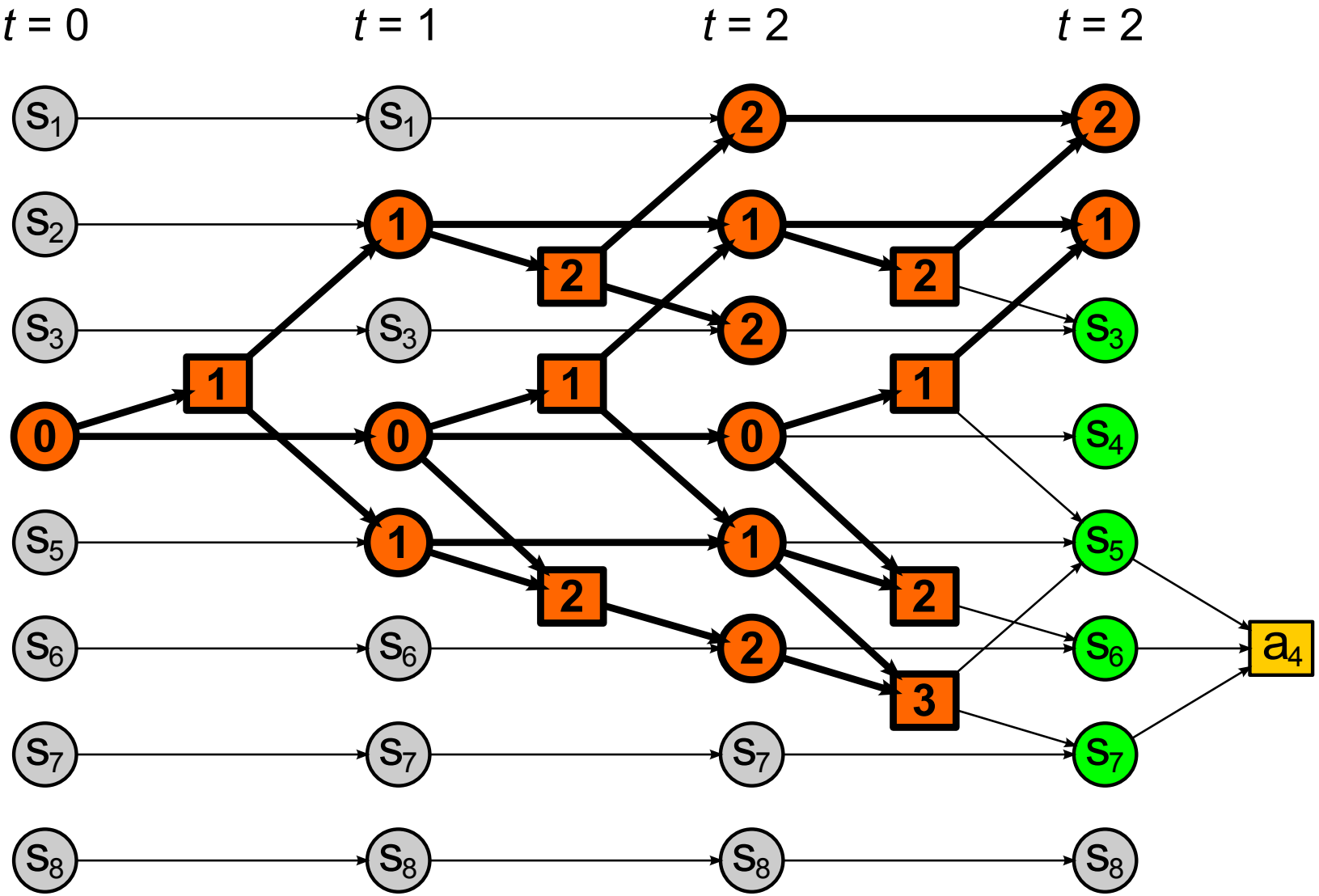
Heuristika h_{max}: Příklad



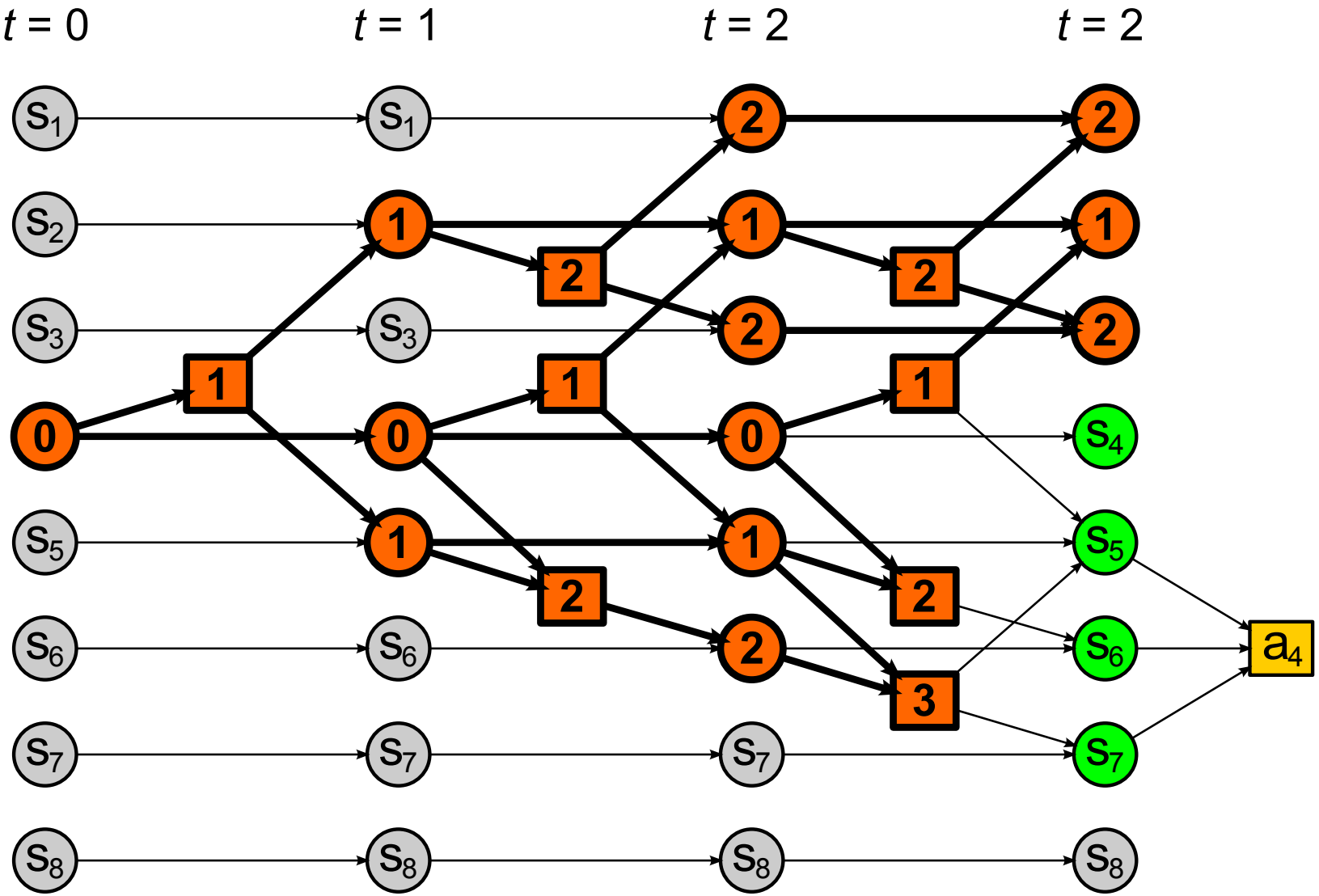
Heuristika h_{max}: Příklad



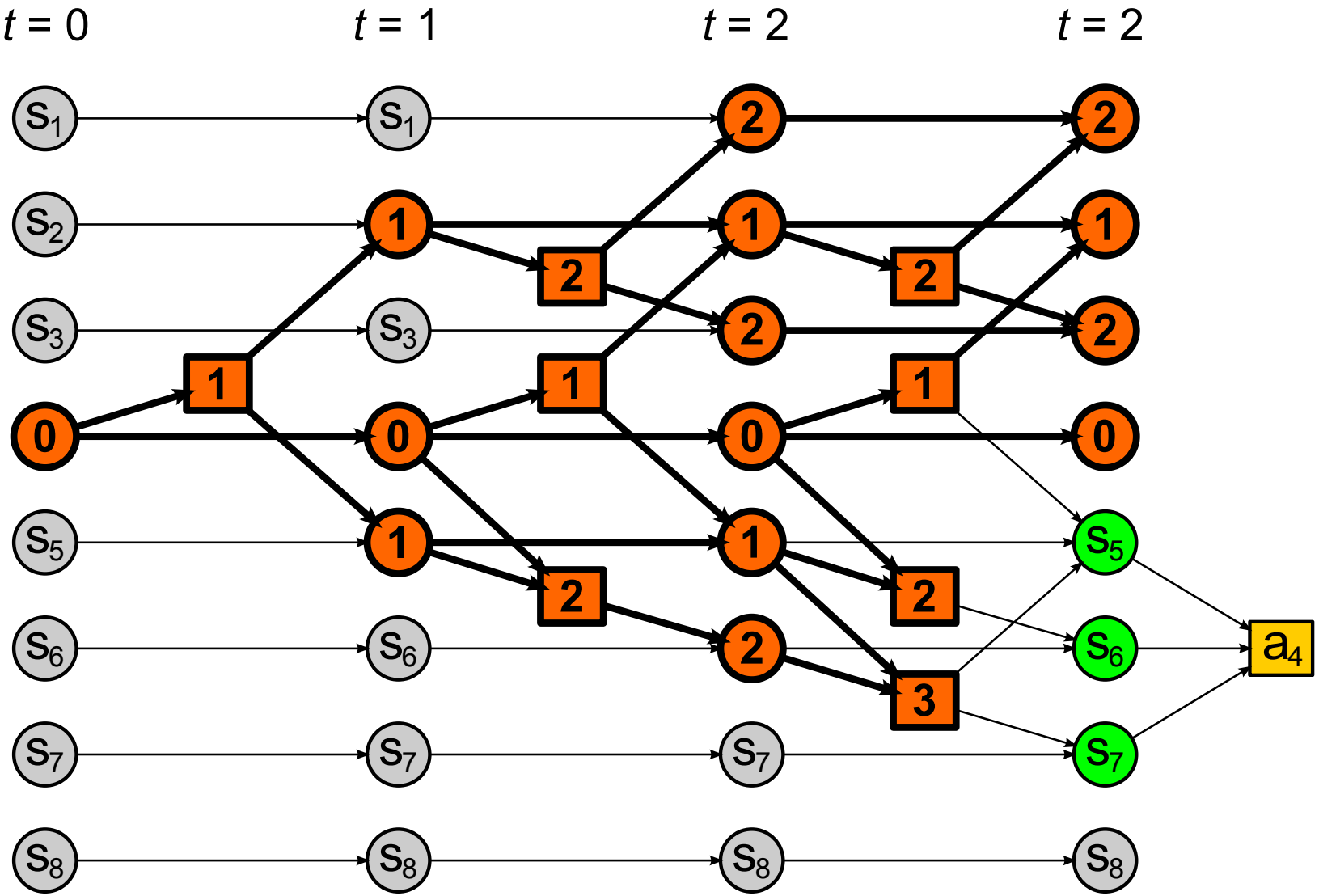
Heuristika h_{max}: Příklad



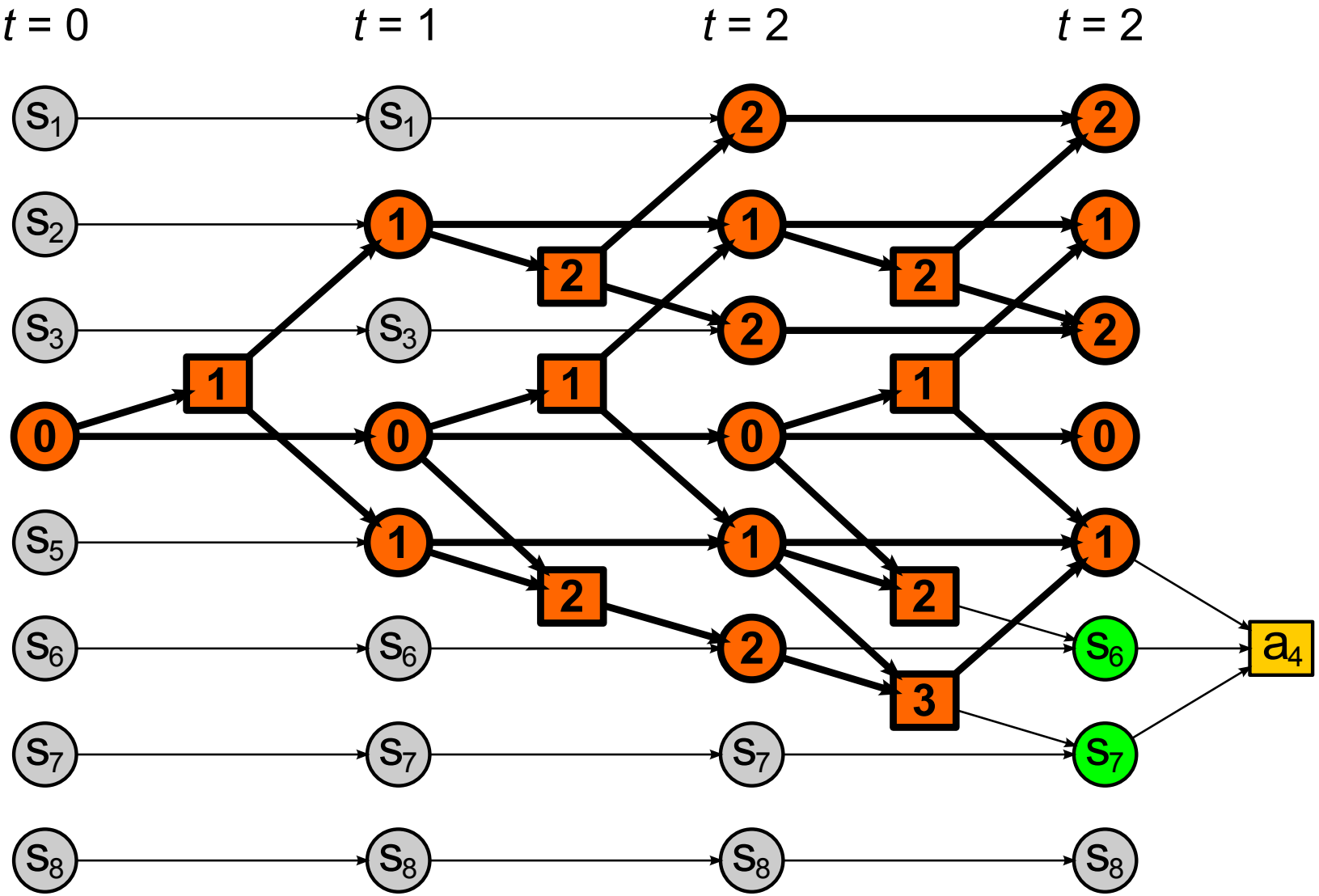
Heuristika h_{max}: Příklad



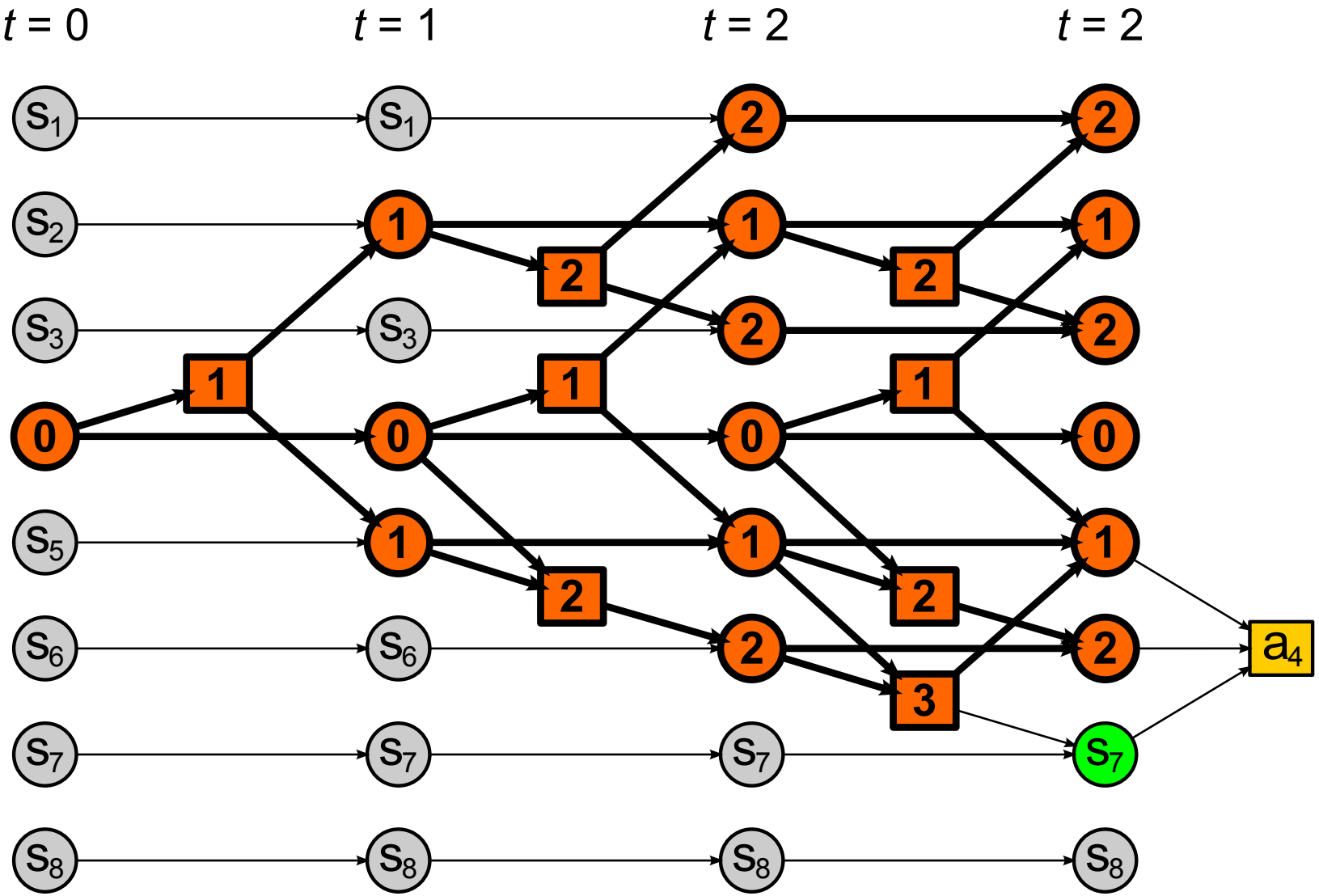
Heuristika h_{max}: Příklad



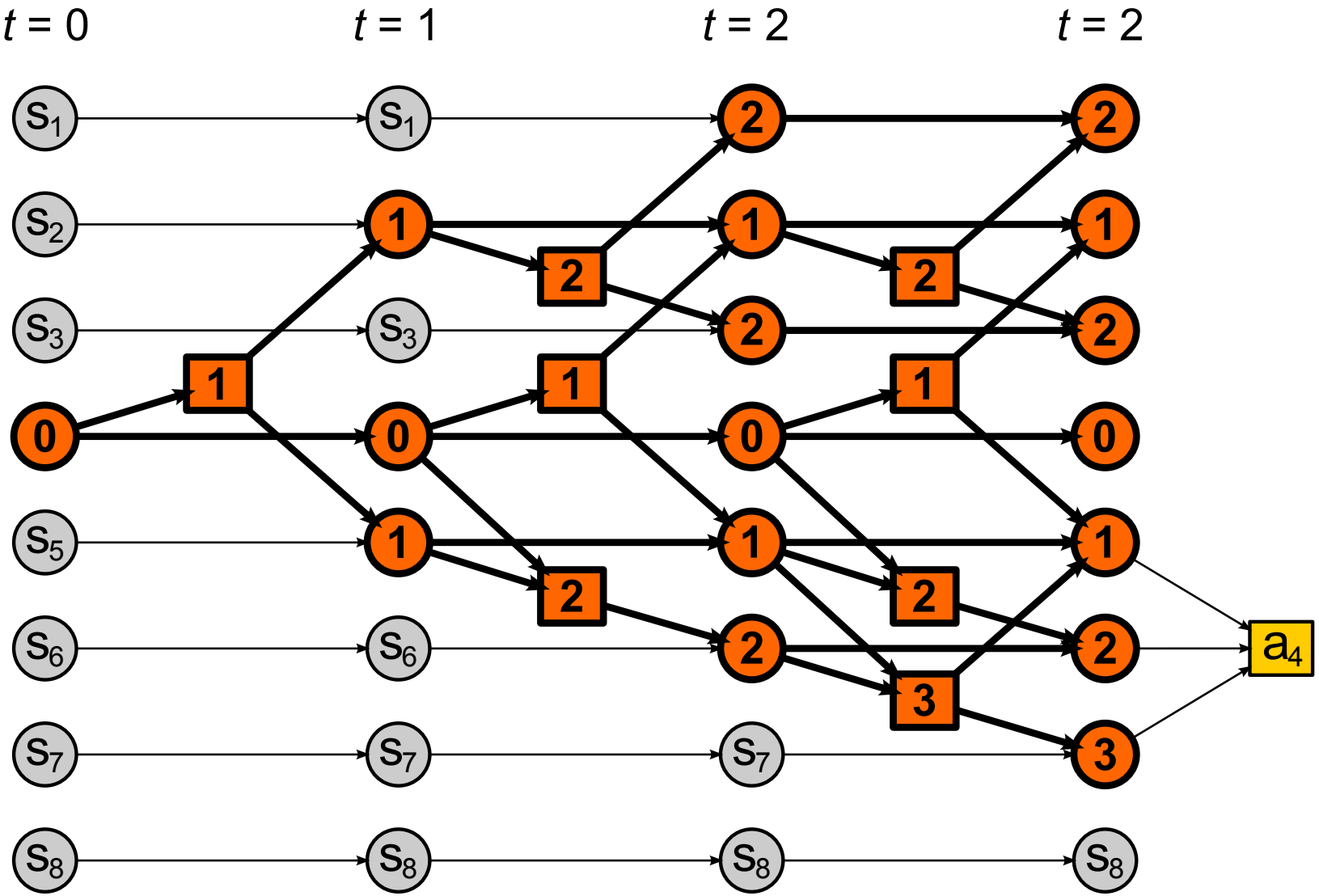
Heuristika h_{max}: Příklad



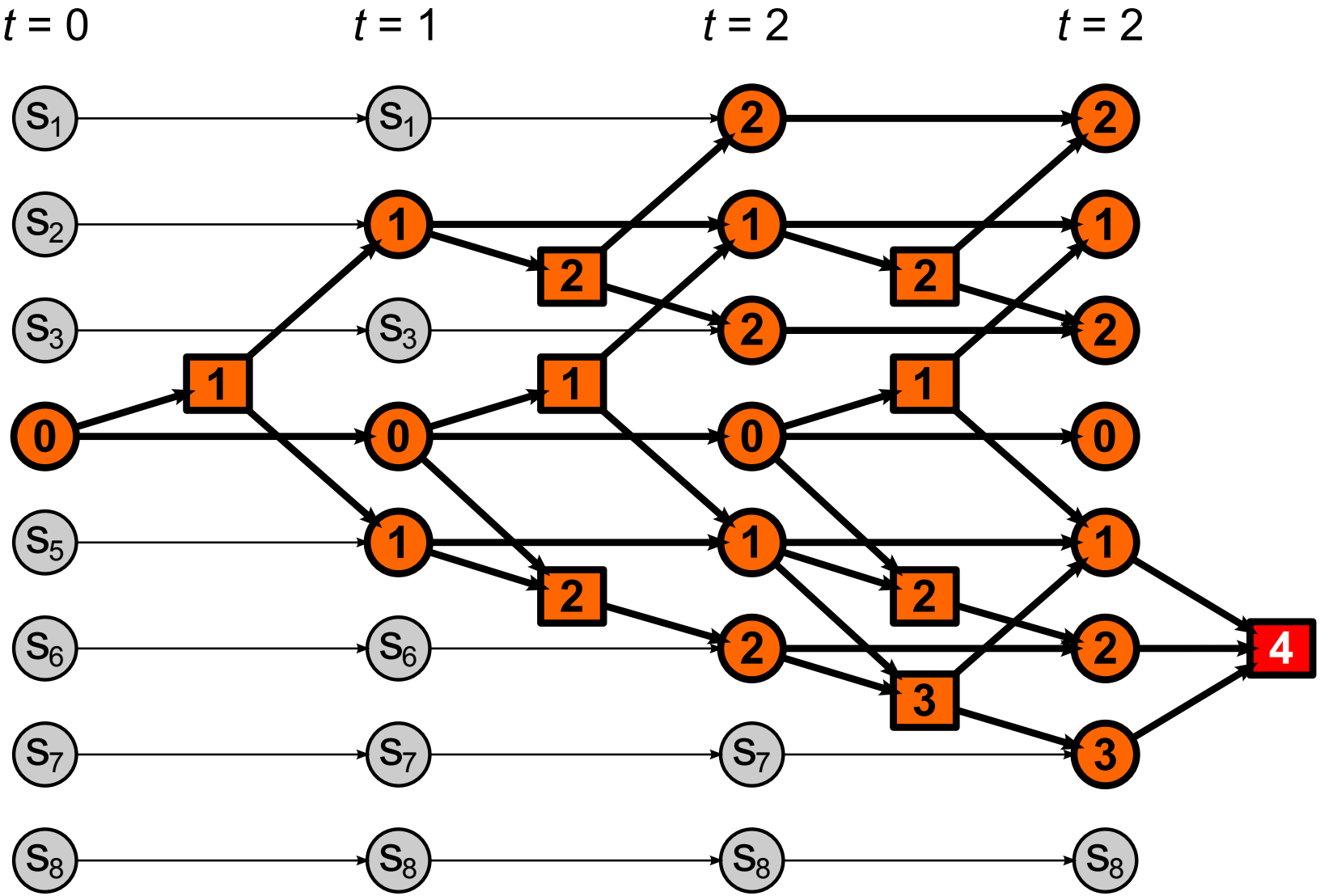
Heuristika h_{max}: Příklad



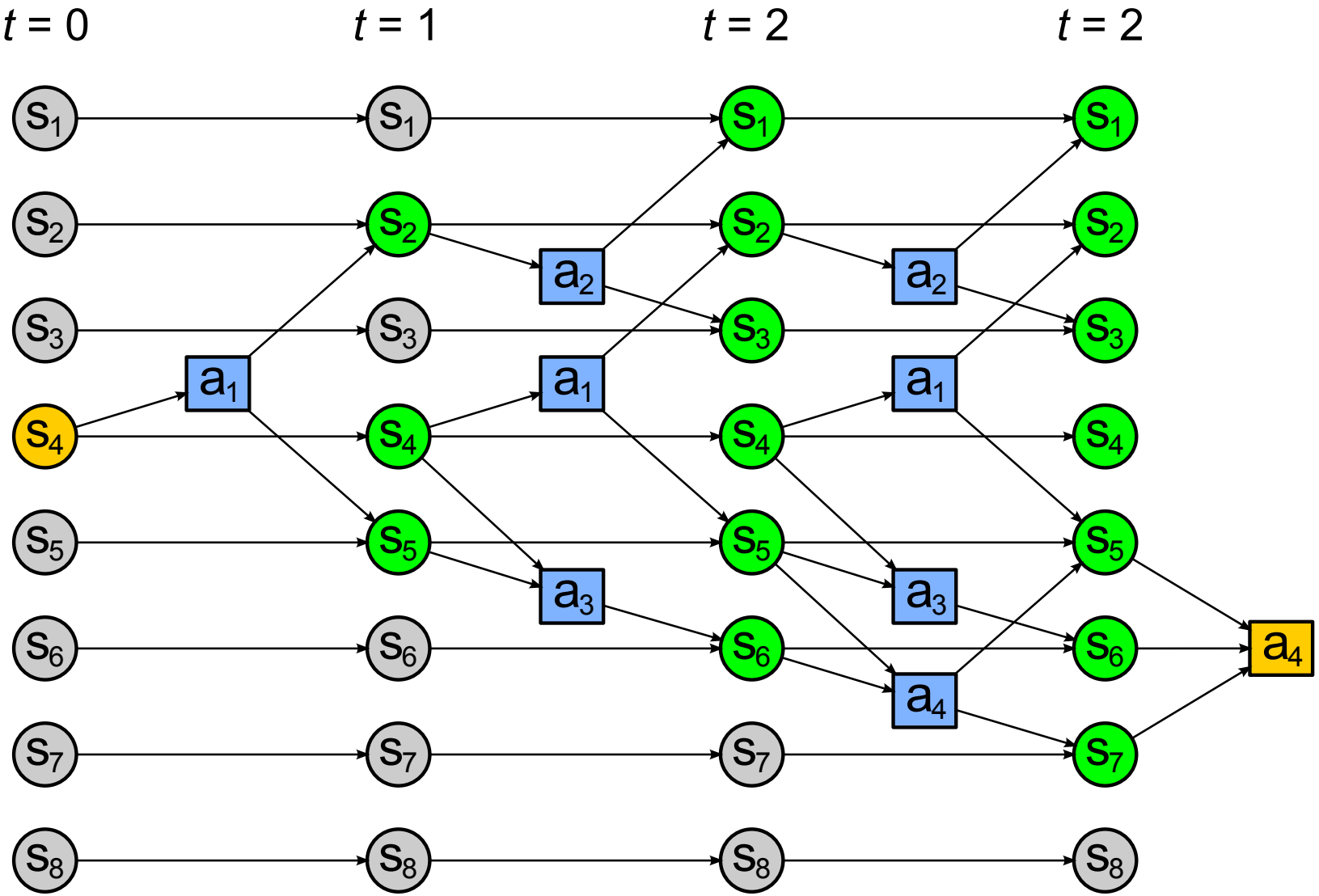
Heuristika h_{max}: Příklad



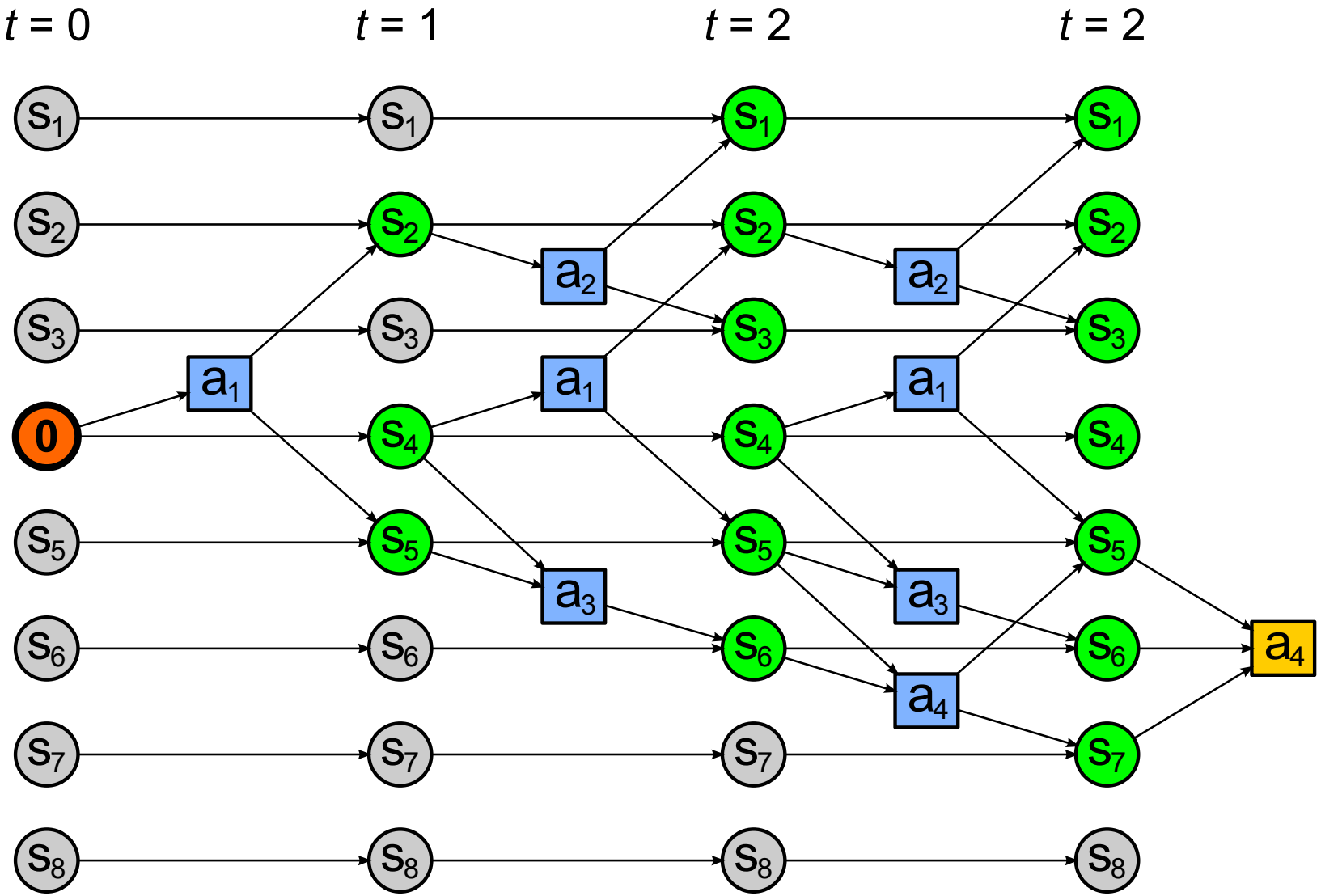
Heuristika h_{max}: Příklad



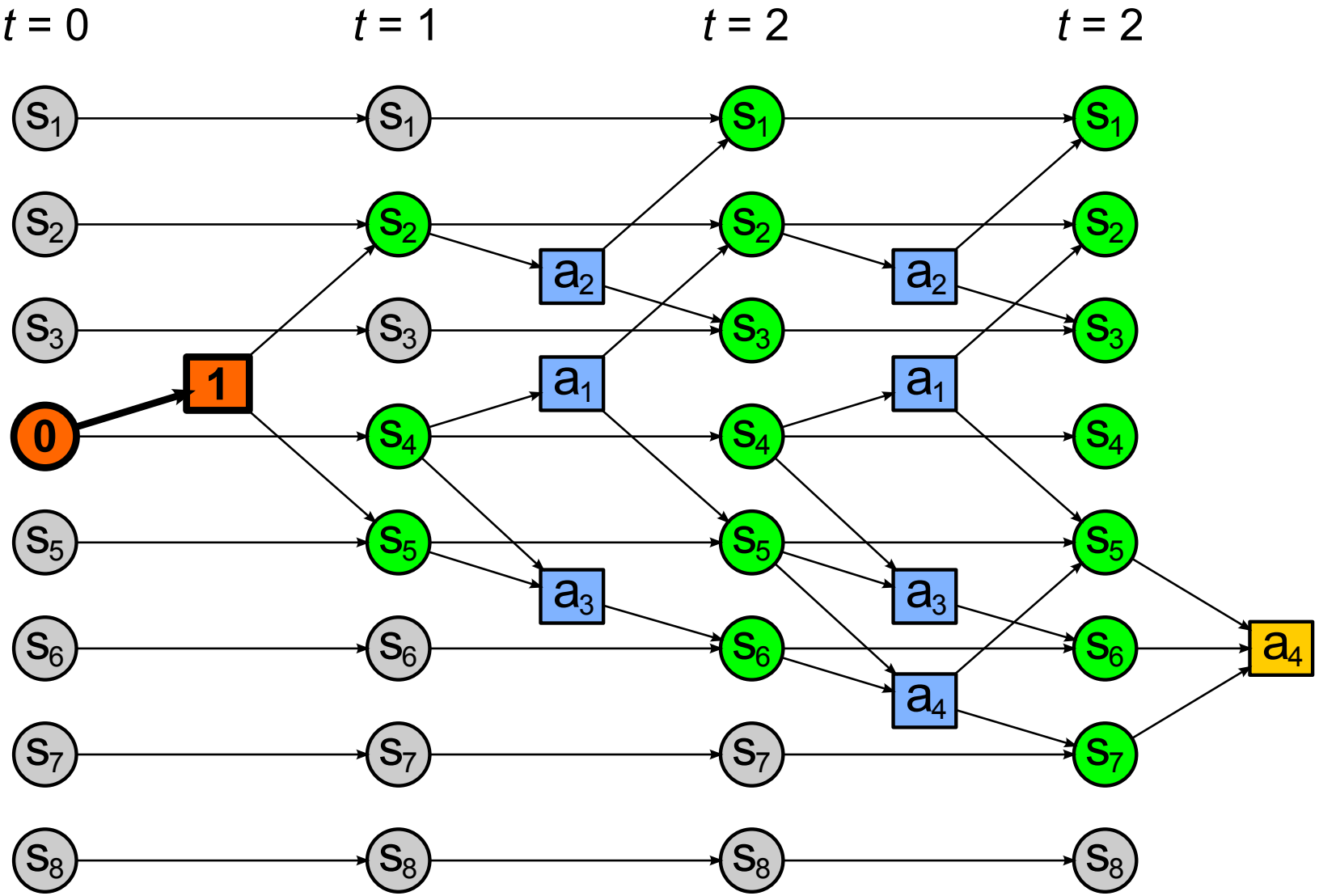
Heuristika h_{add}: Příklad



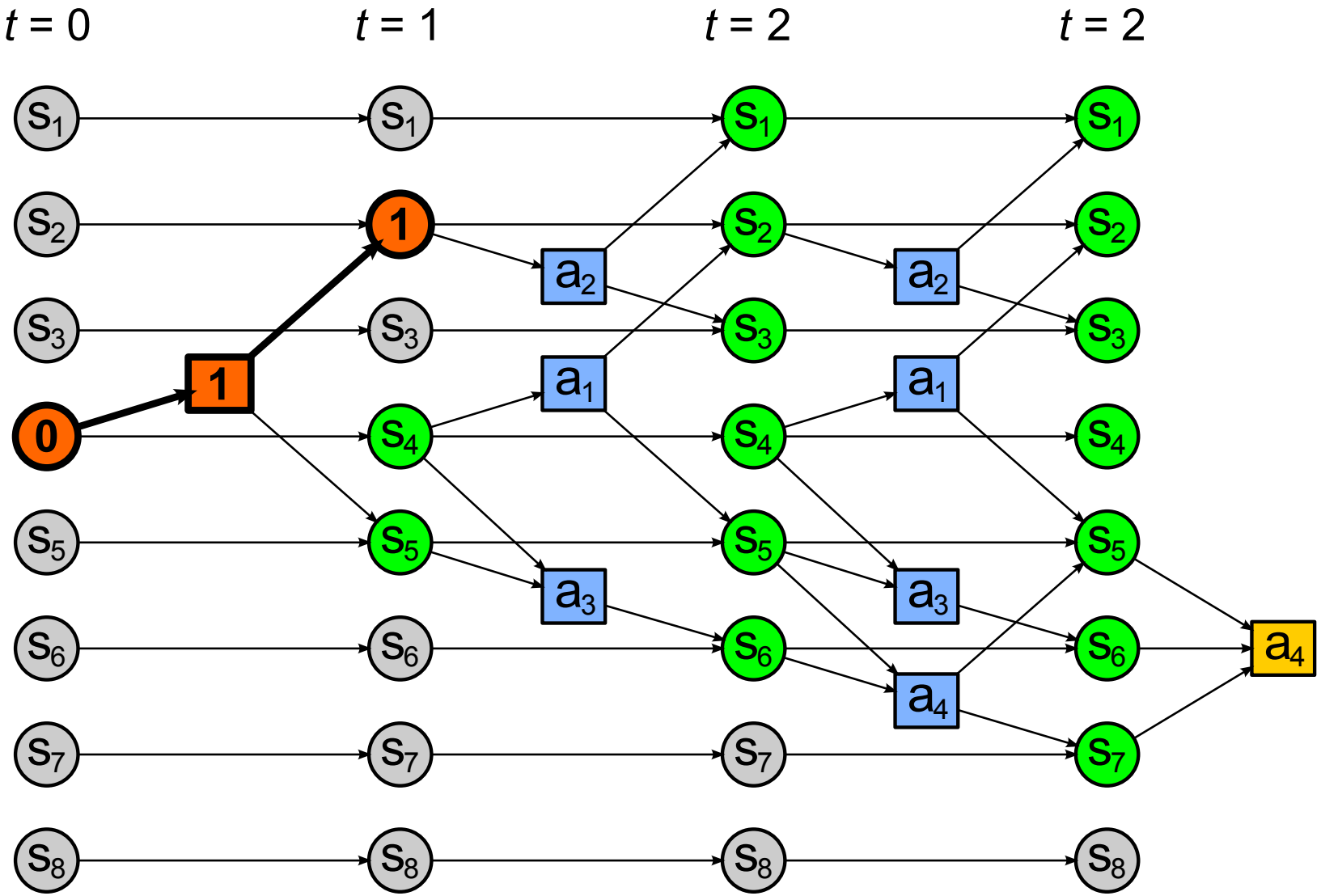
Heuristika h_{add}: Příklad



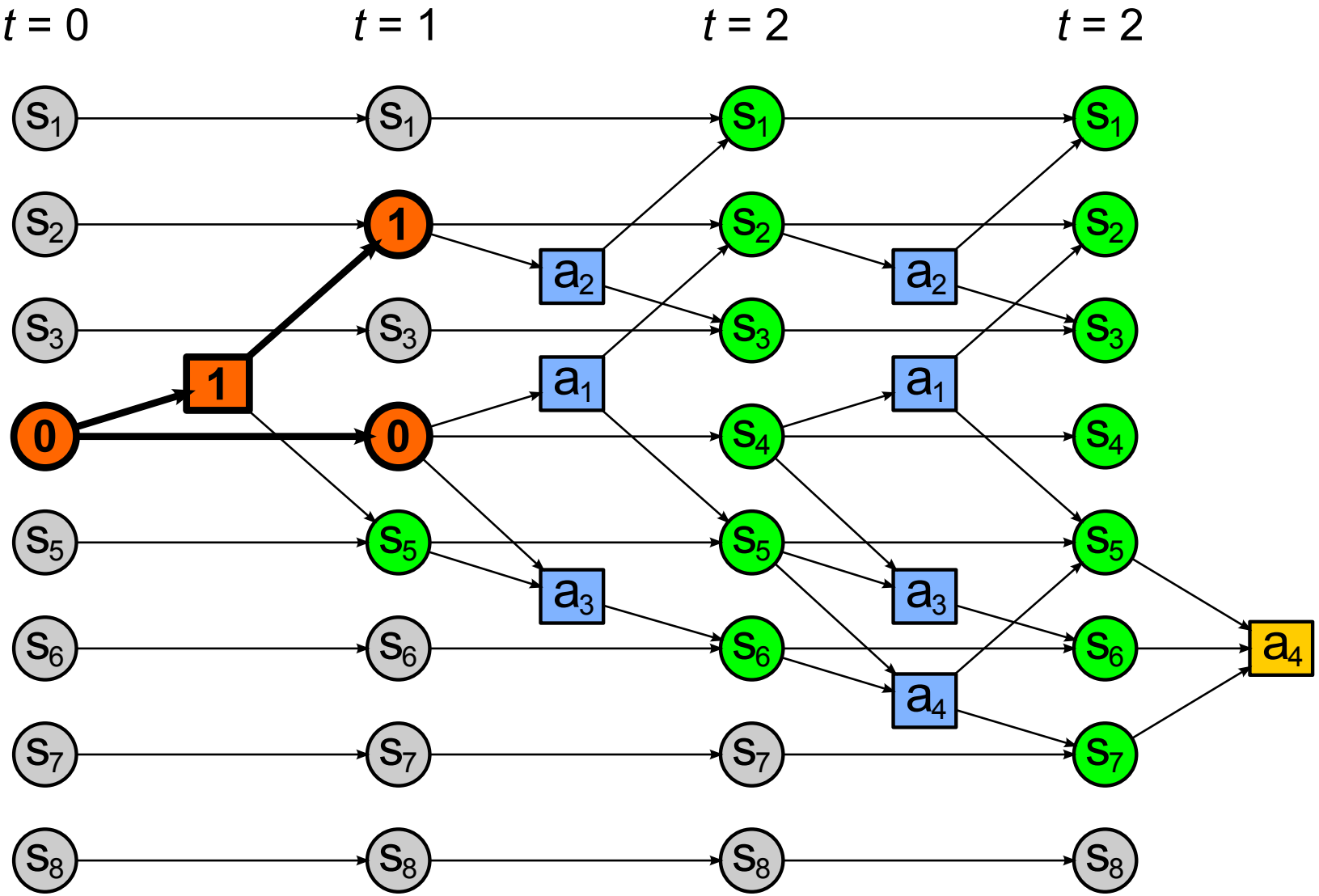
Heuristika h_{add}: Příklad



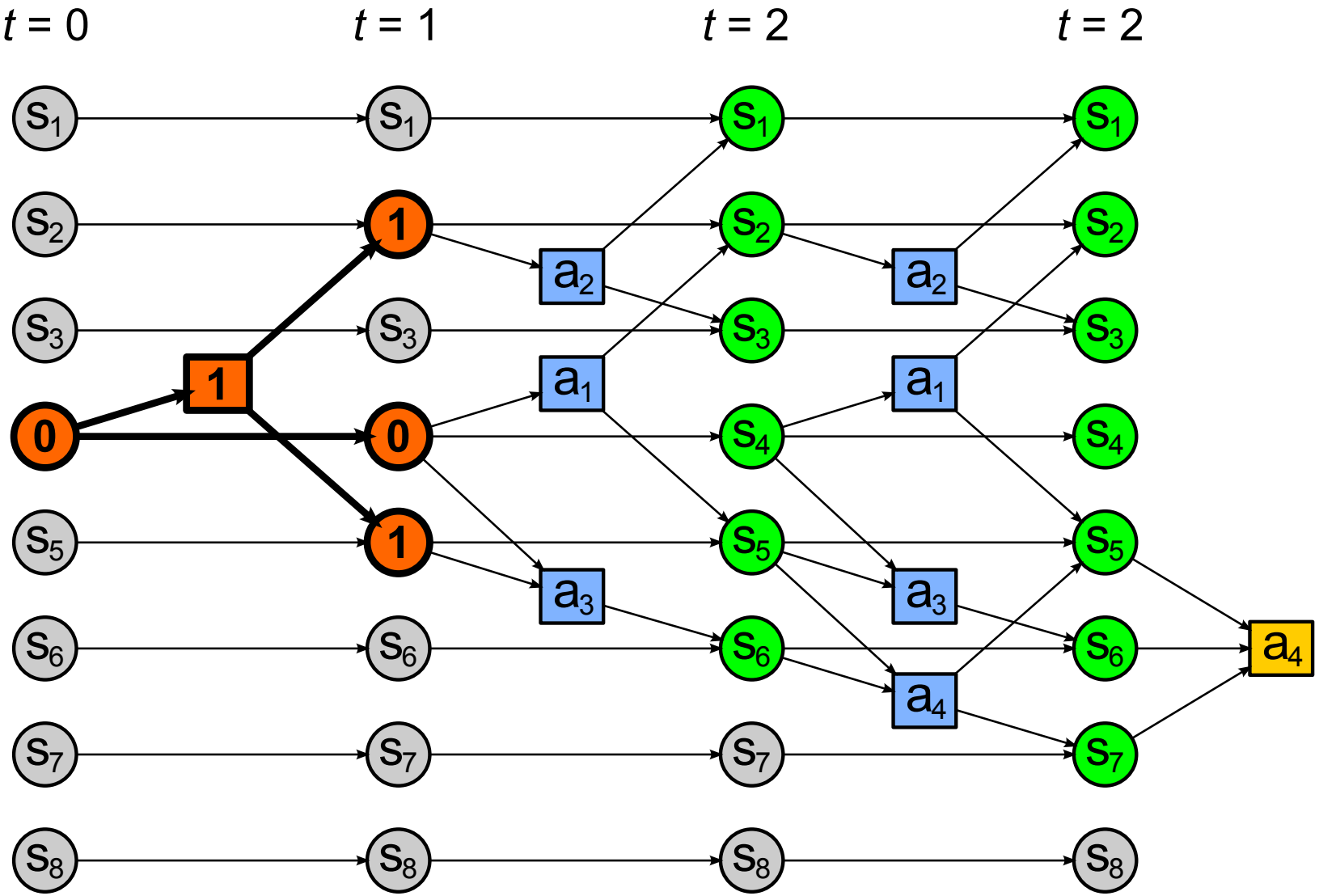
Heuristika h_{add}: Příklad



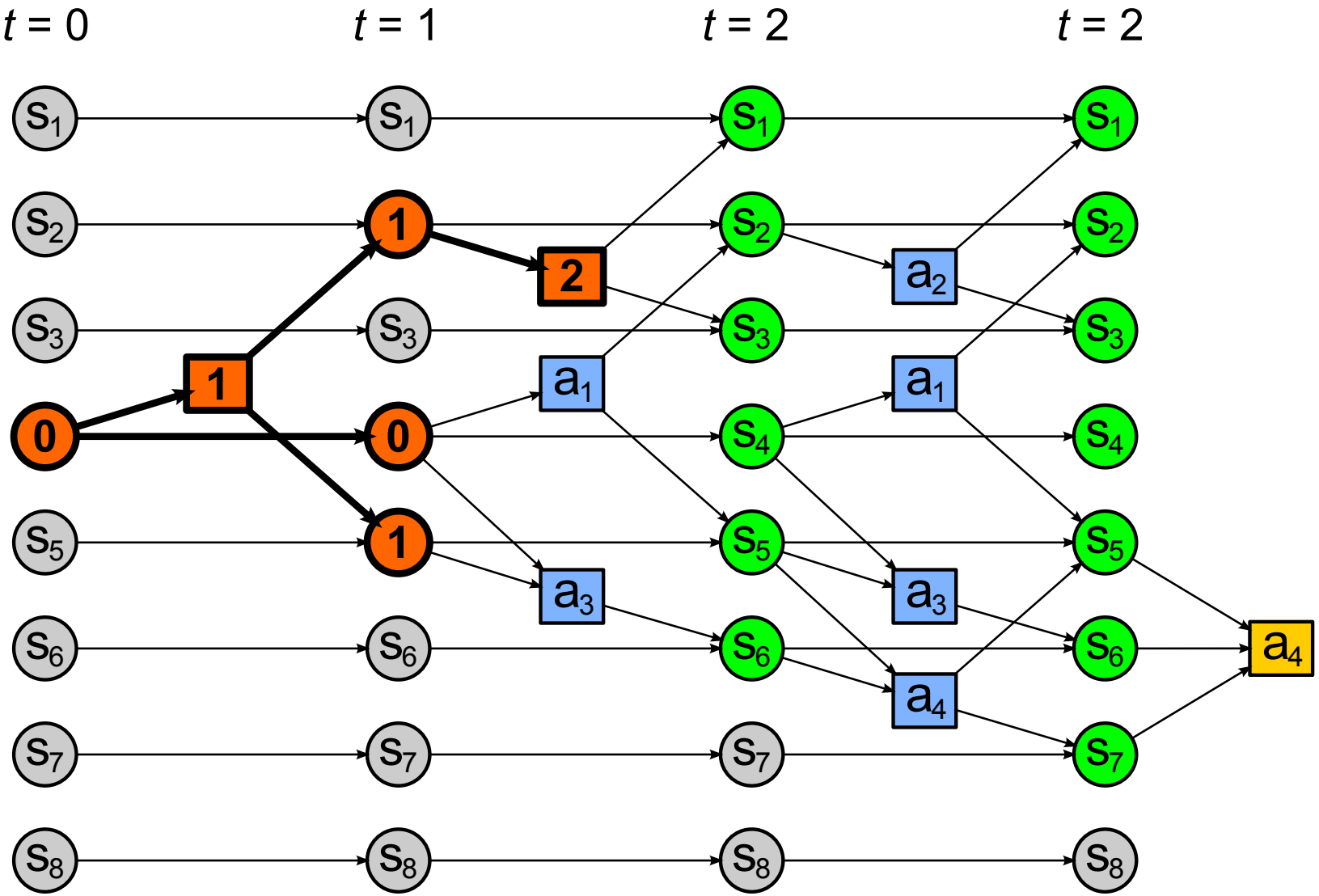
Heuristika h_{add}: Příklad



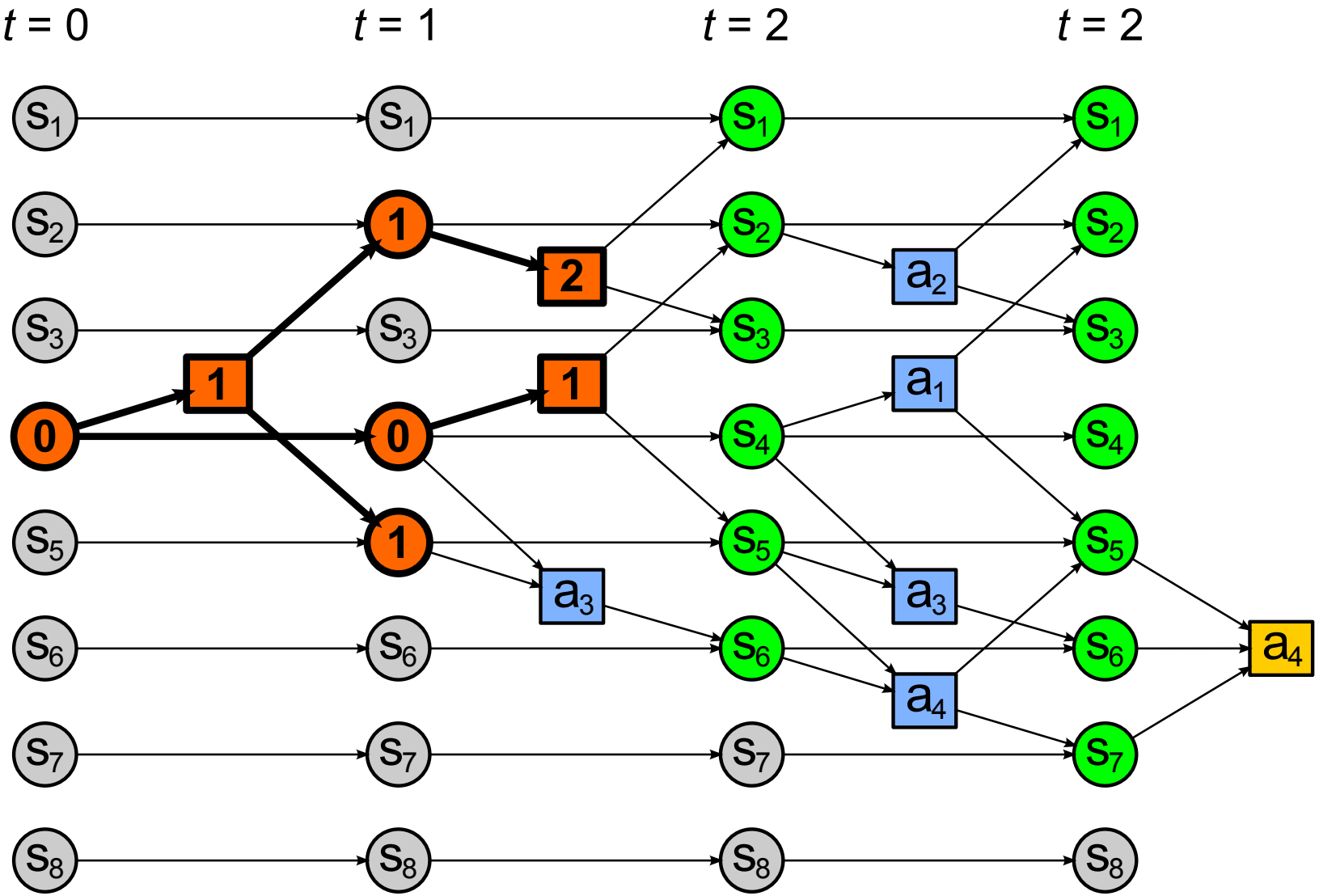
Heuristika h_{add}: Příklad



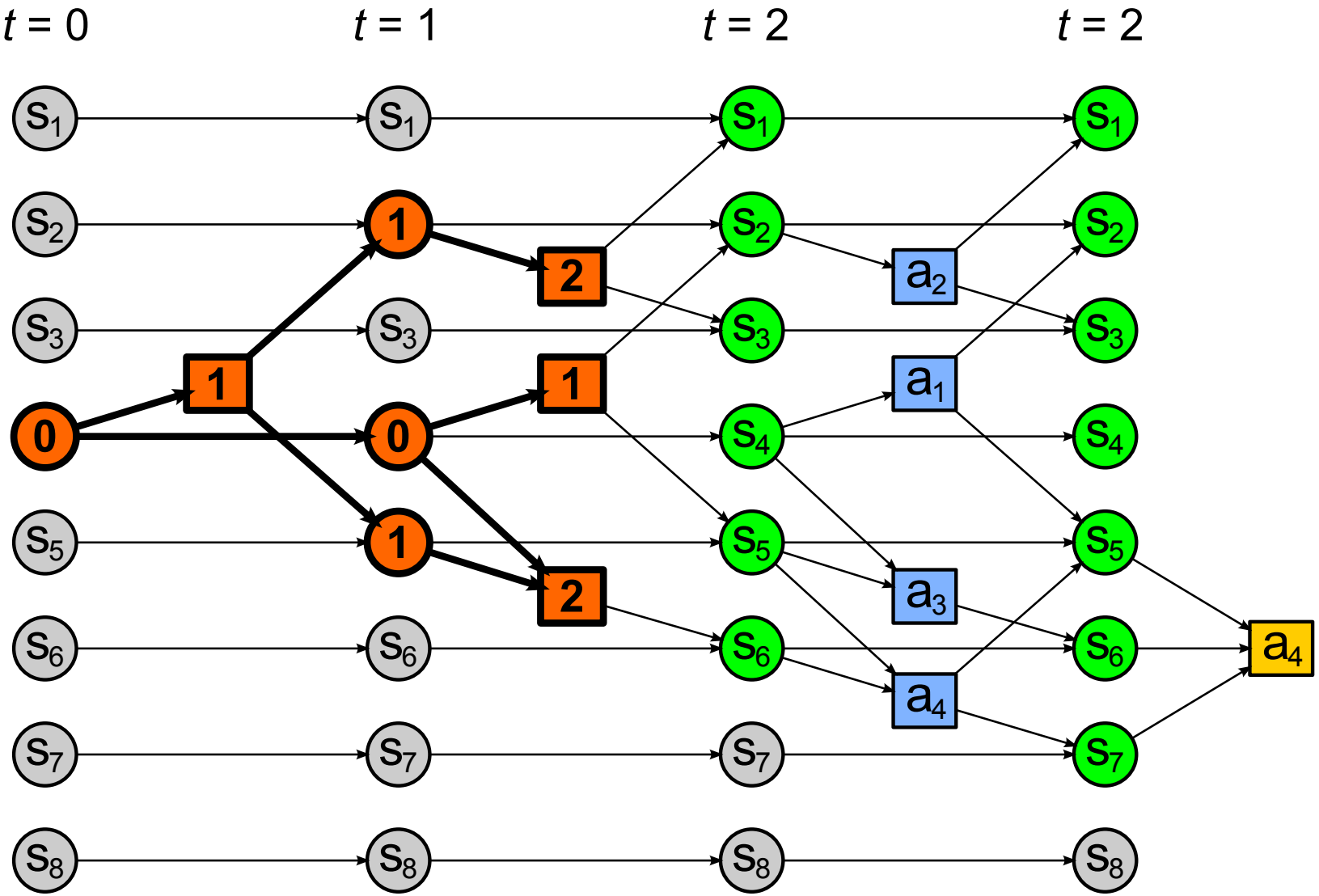
Heuristika h_{add}: Příklad



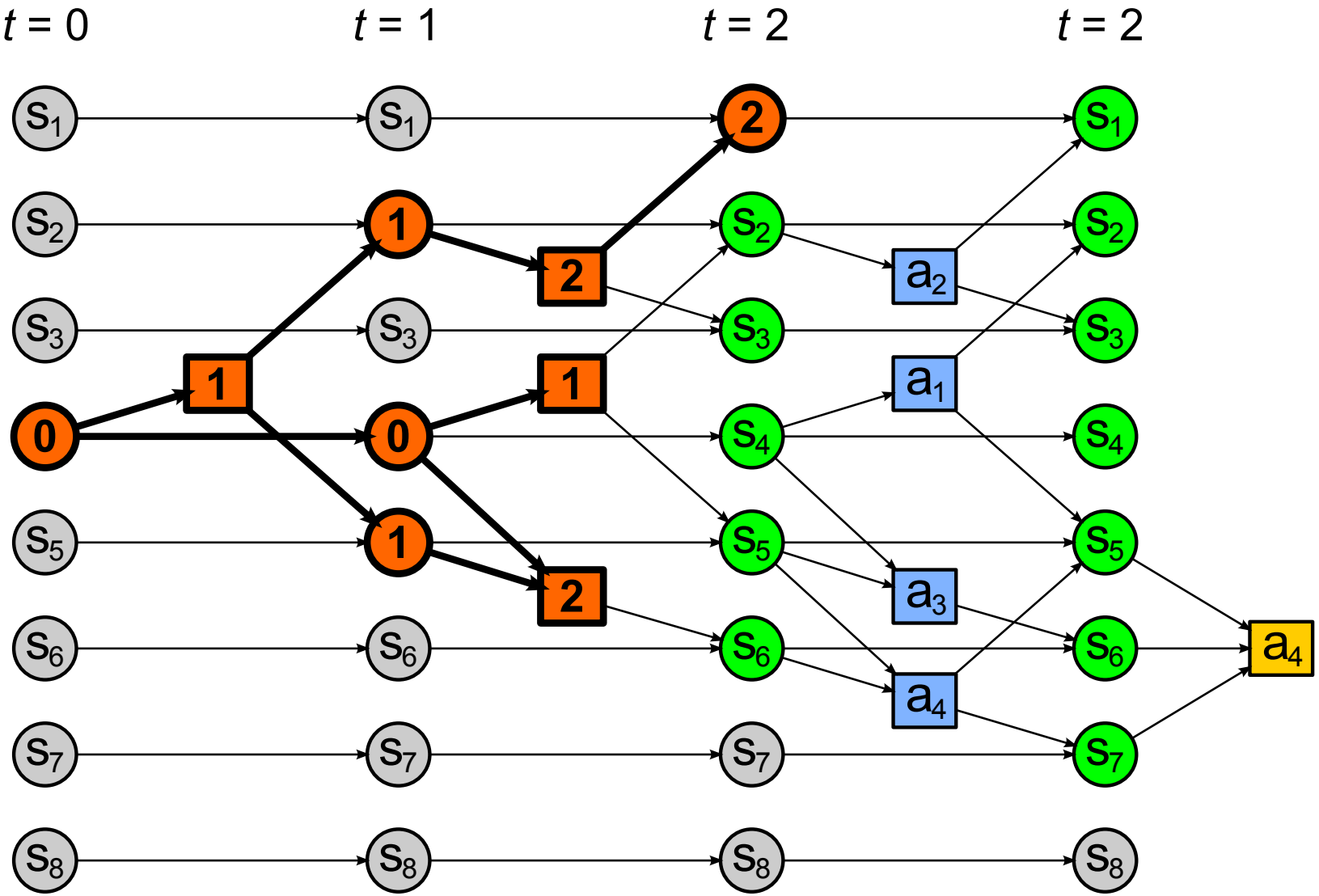
Heuristika h_{add}: Příklad



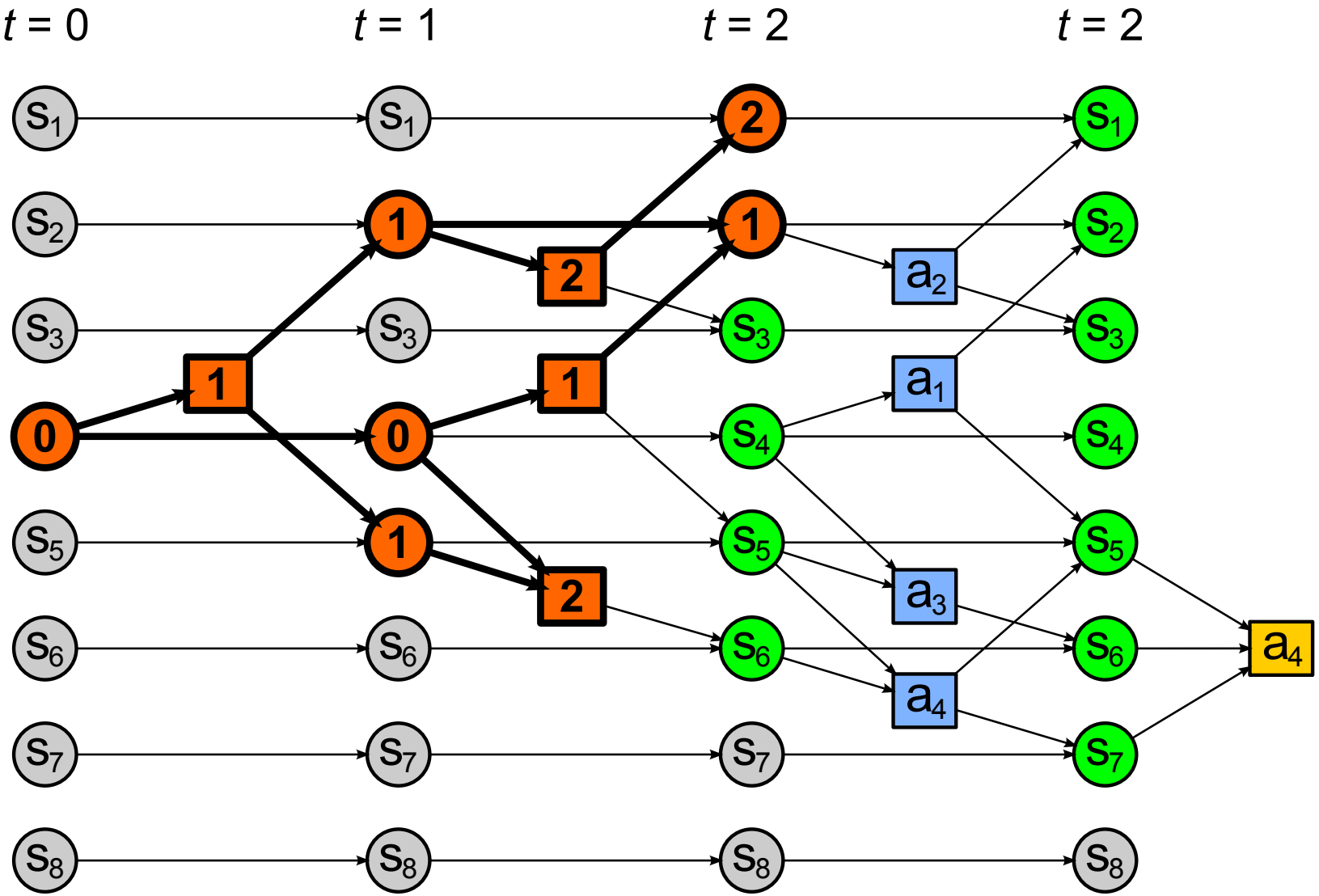
Heuristika h_{add}: Příklad



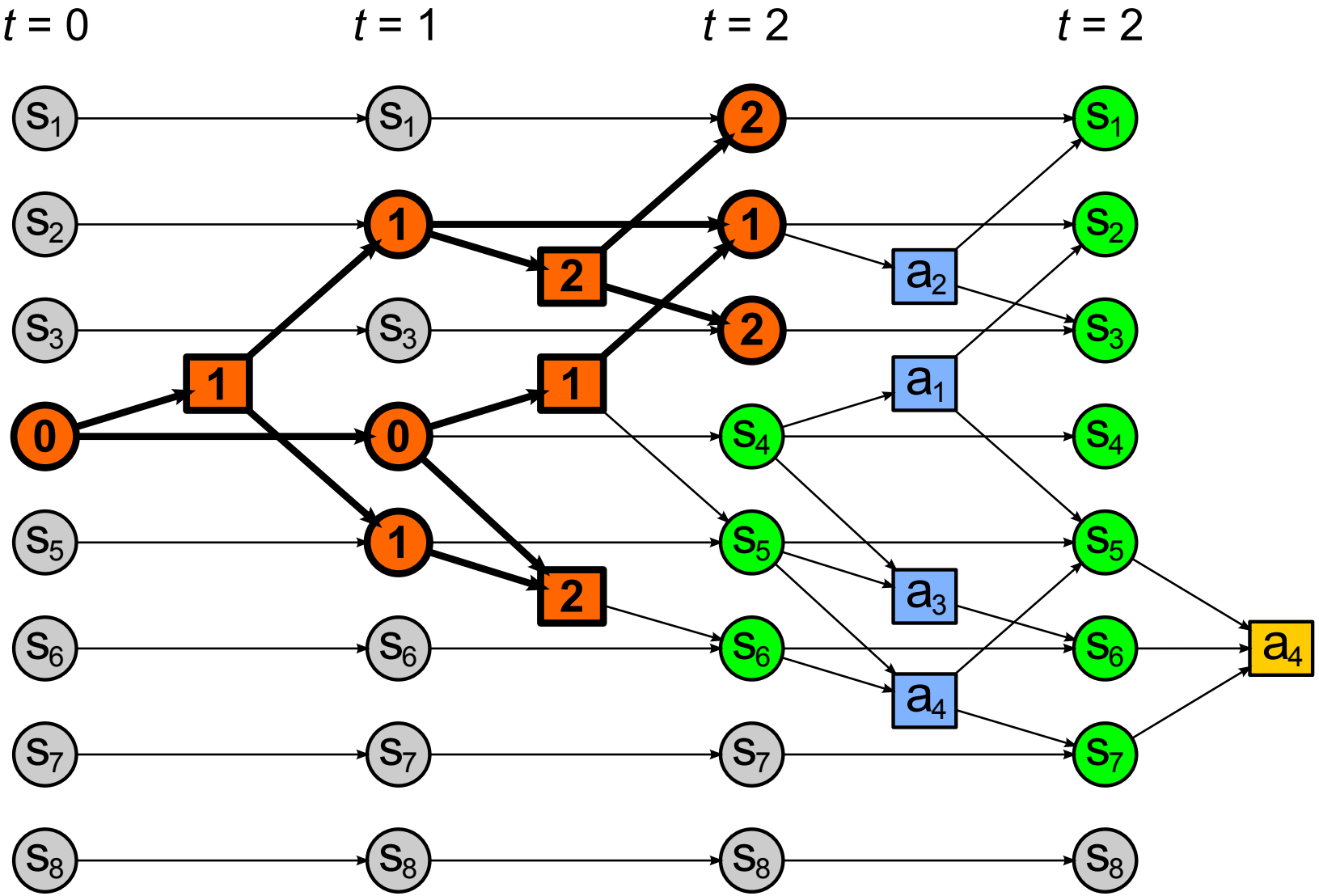
Heuristika h_{add}: Příklad



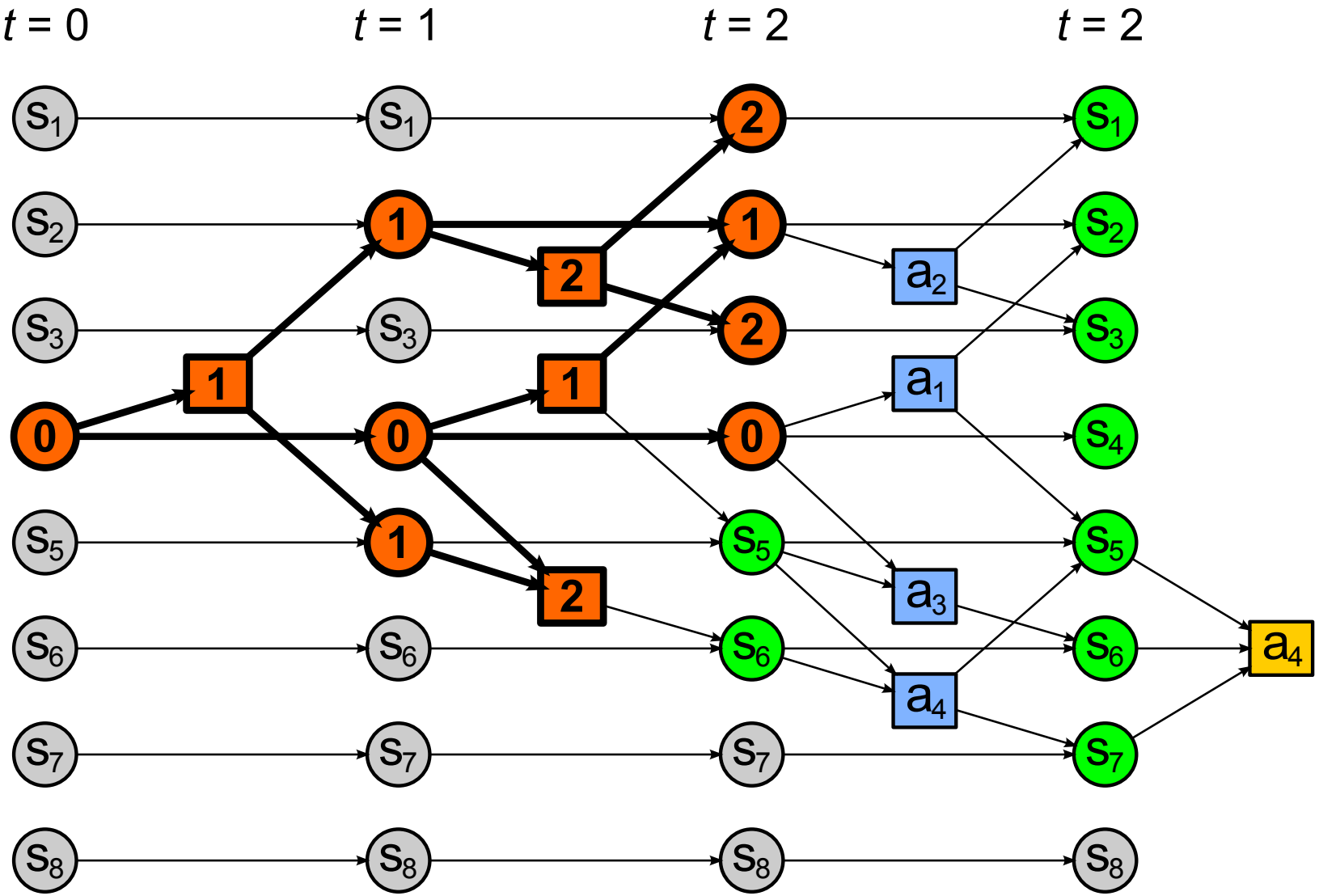
Heuristika h_{add}: Příklad



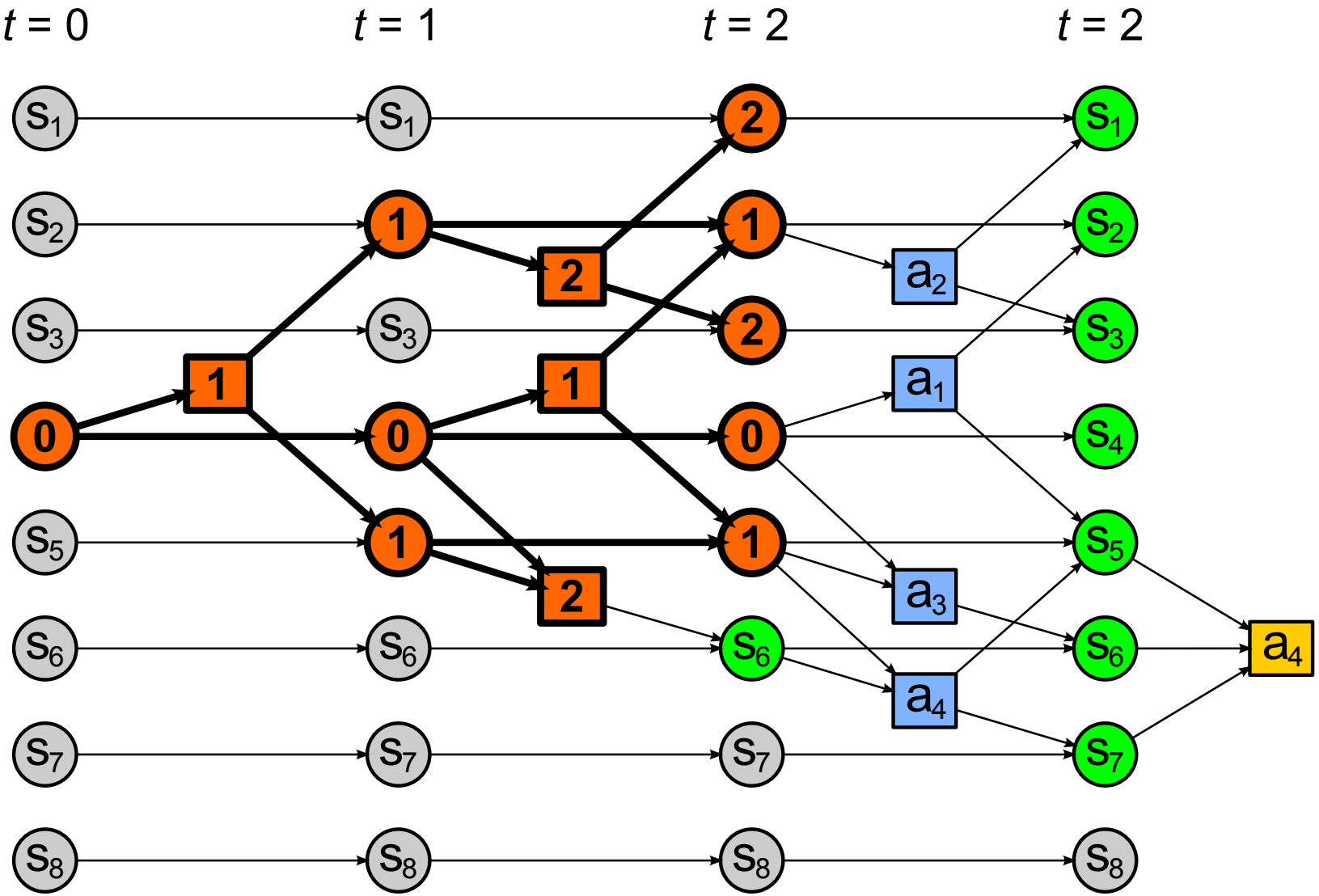
Heuristika h_{add}: Příklad



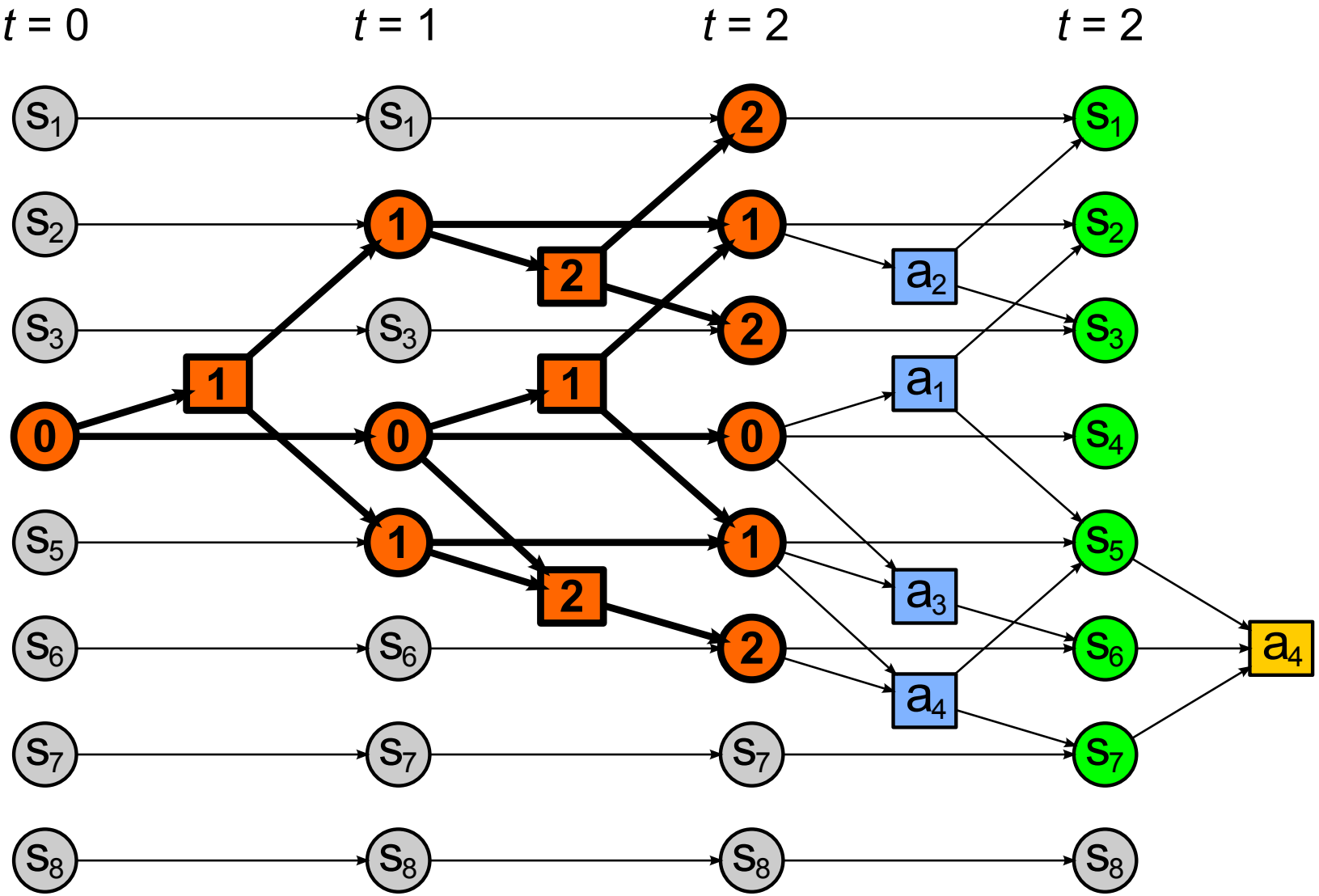
Heuristika h_{add}: Příklad



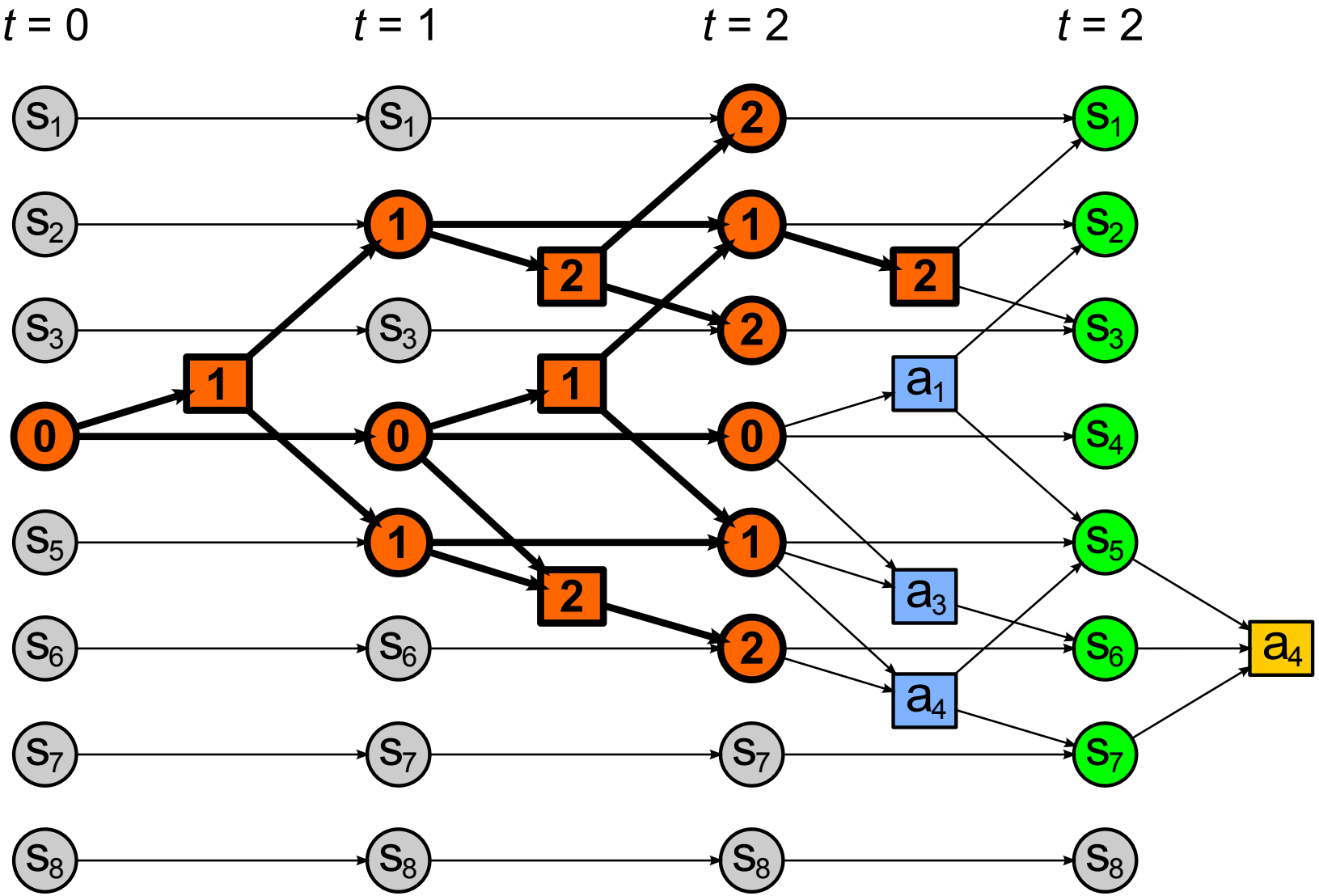
Heuristika h_{add}: Příklad



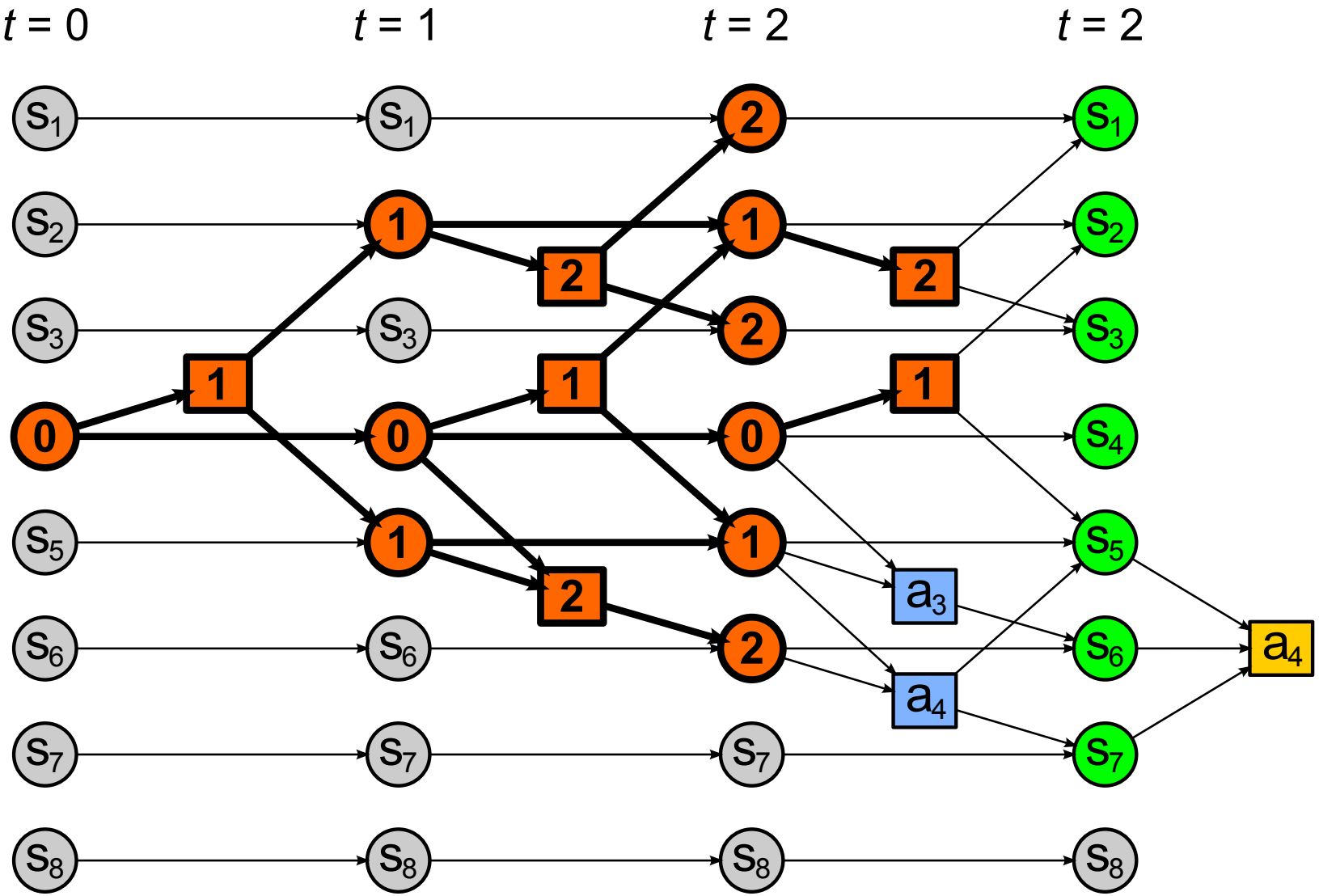
Heuristika h_{add}: Příklad



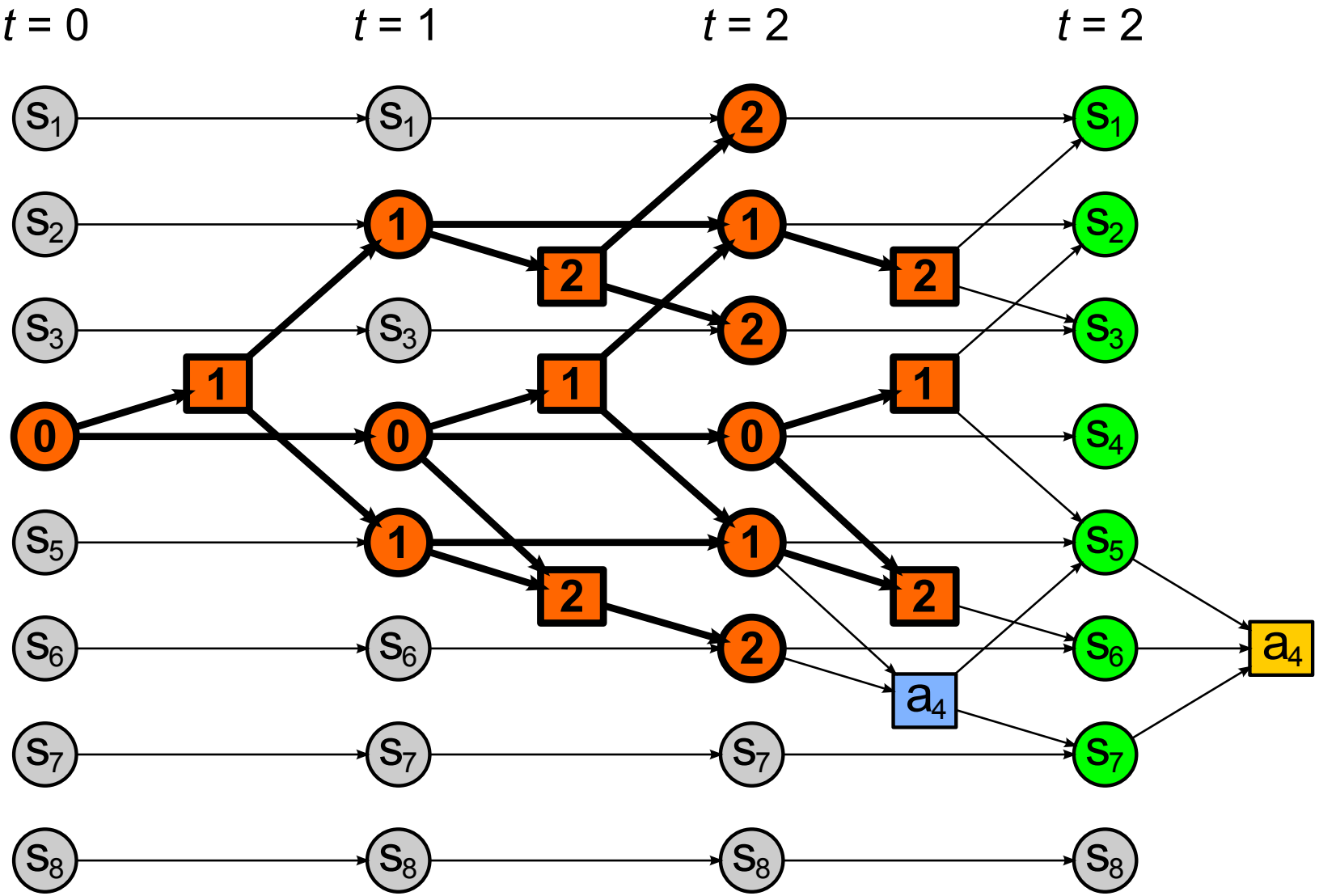
Heuristika h_{add}: Příklad



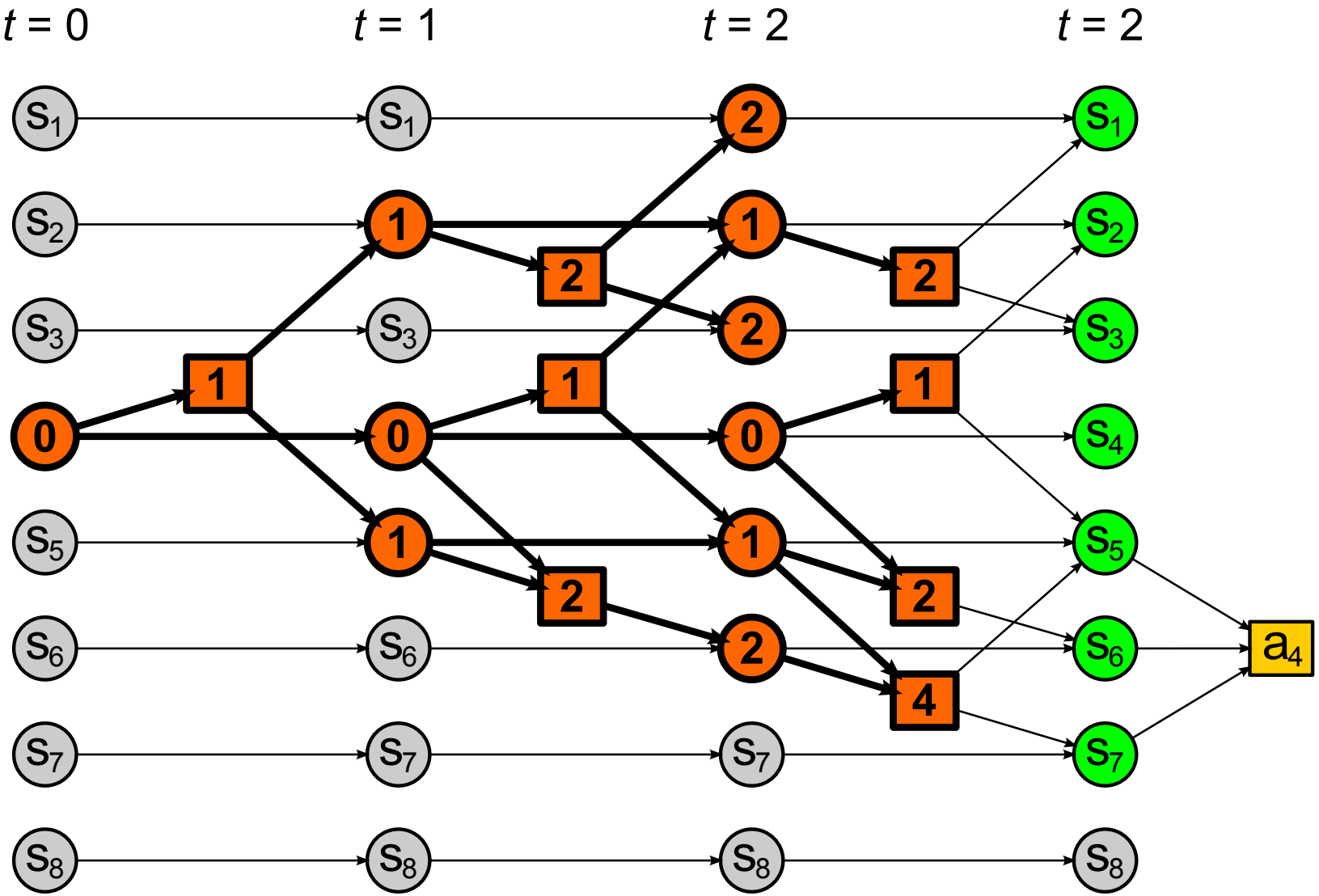
Heuristika h_{add}: Příklad



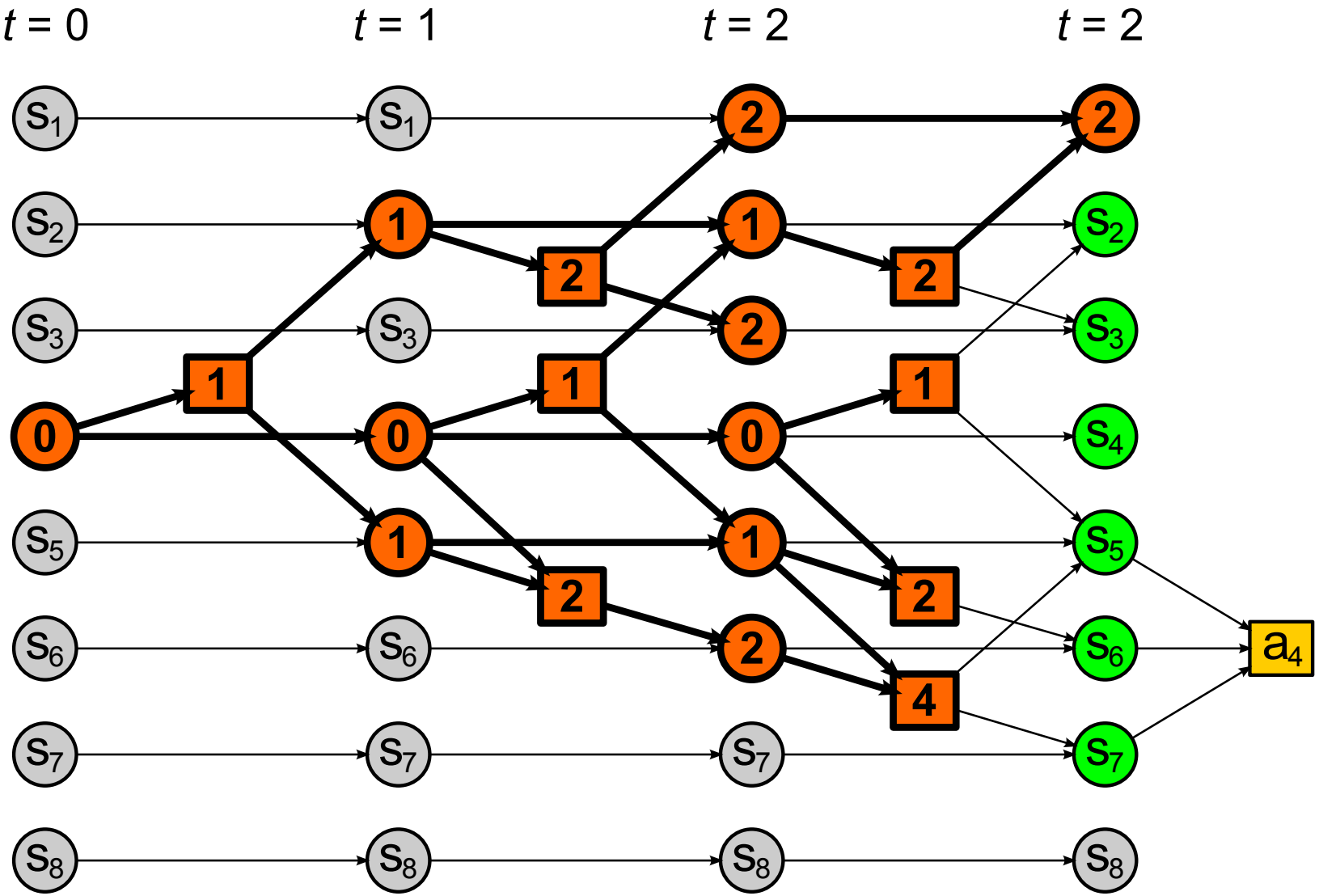
Heuristika h_{add}: Příklad



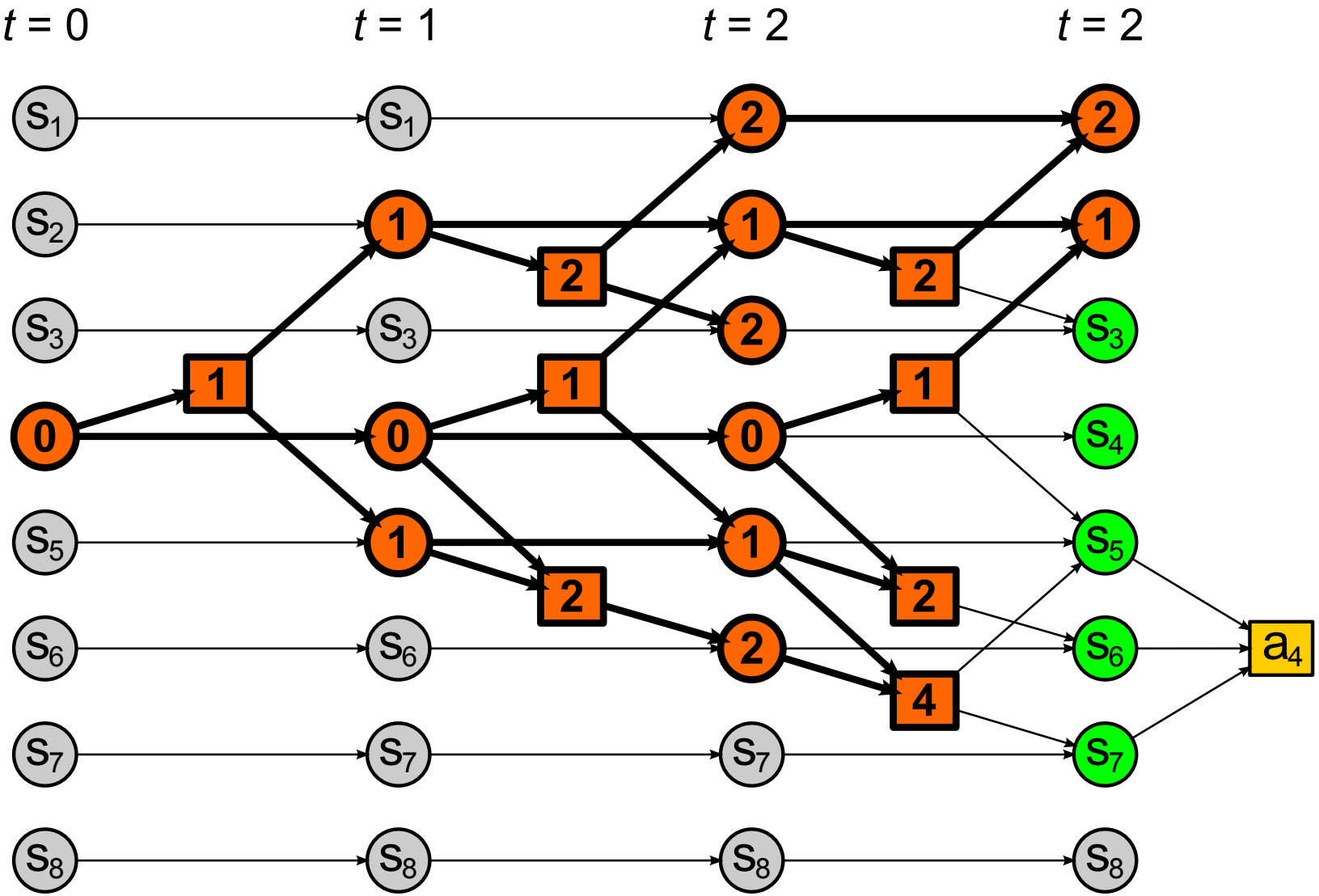
Heuristika h_{add}: Příklad



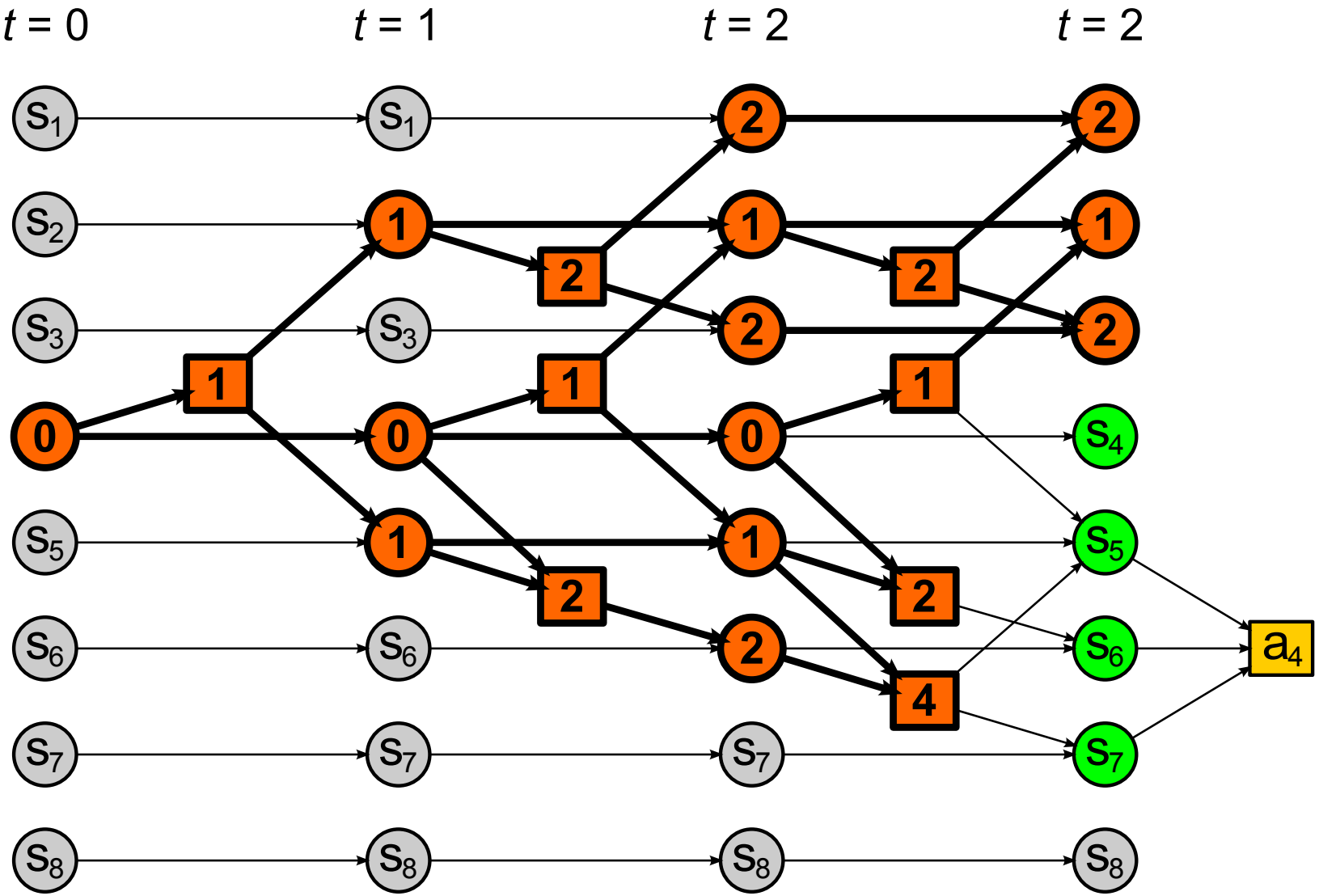
Heuristika h_{add}: Příklad



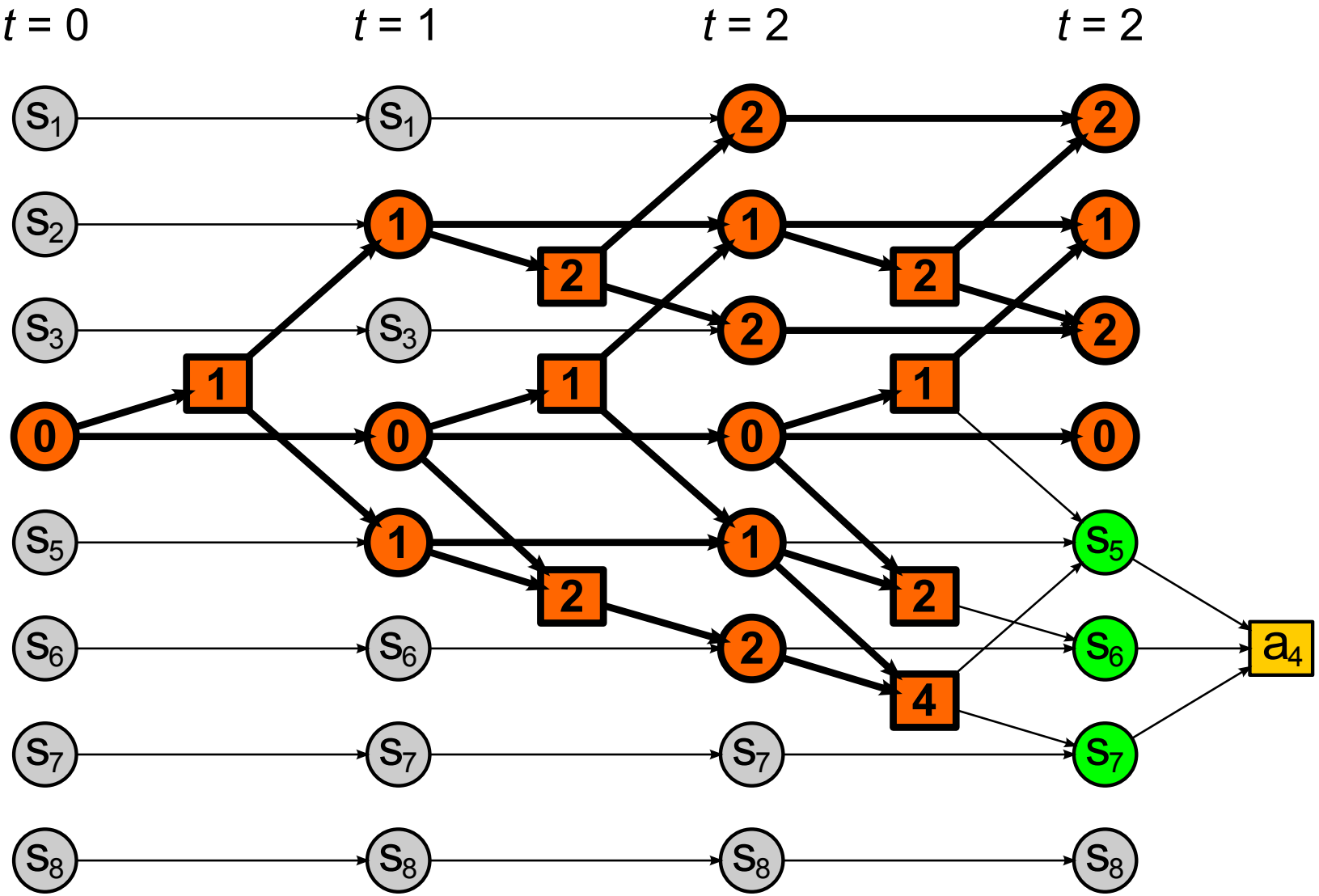
Heuristika h_{add}: Příklad



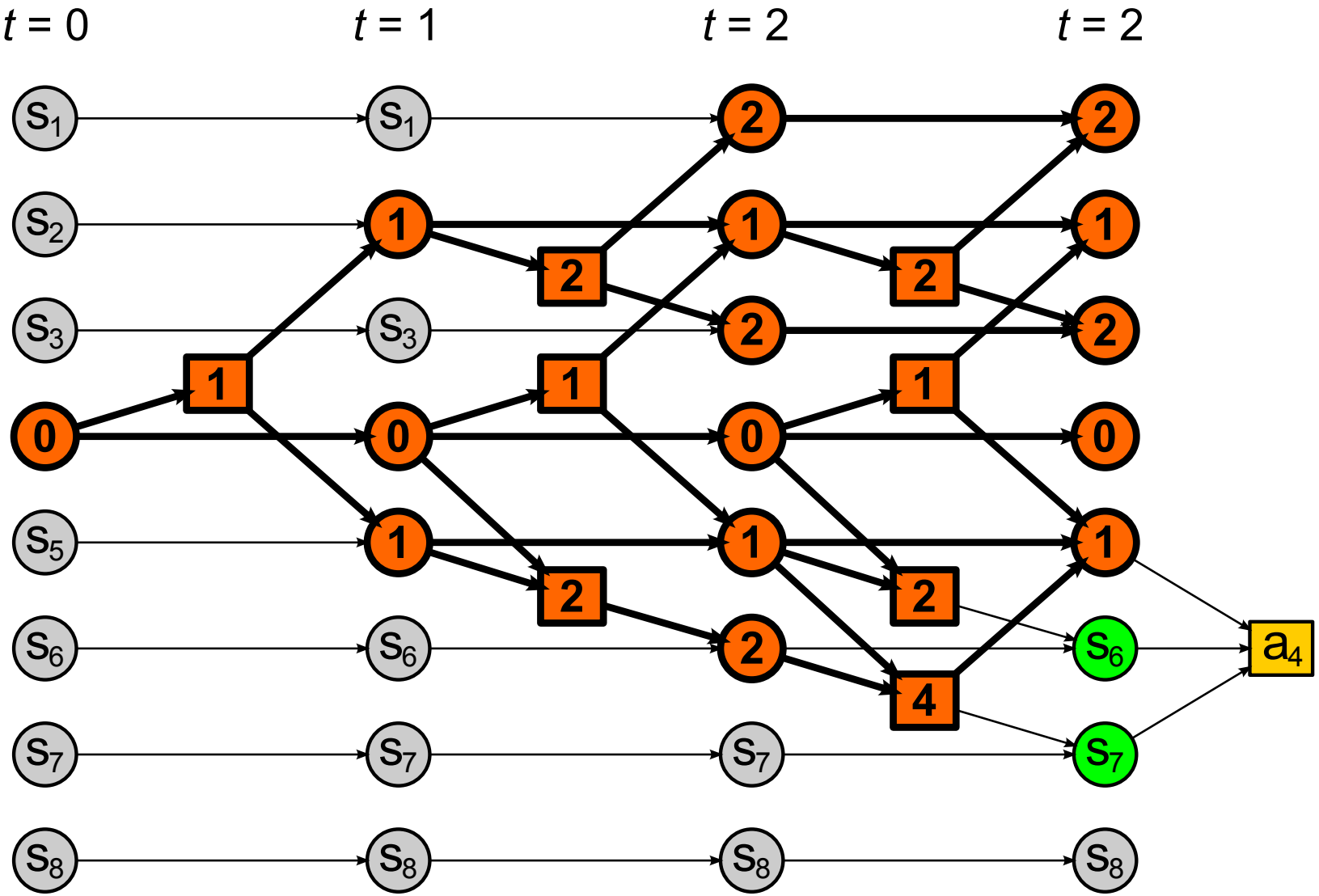
Heuristika h_{add}: Příklad



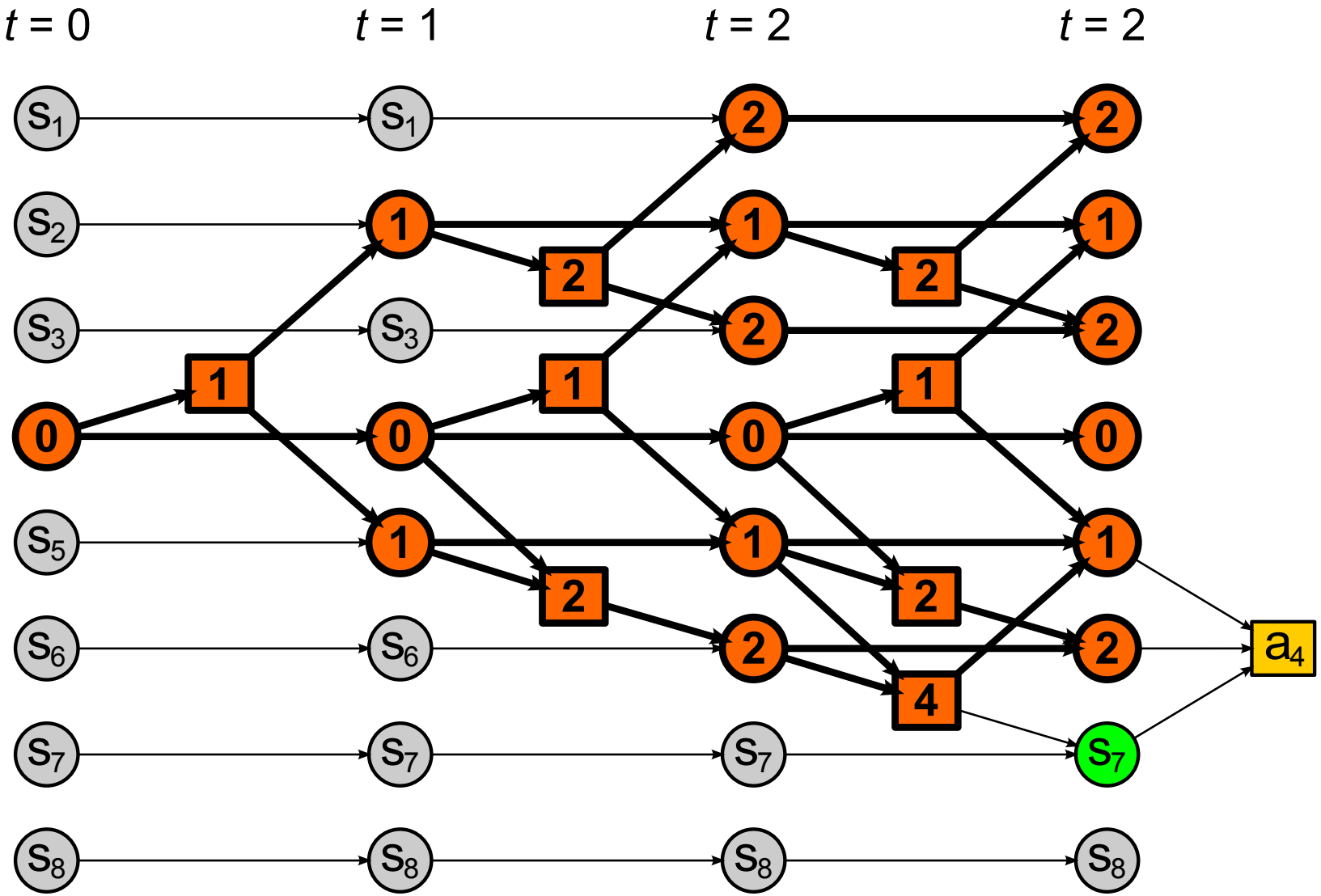
Heuristika h_{add}: Příklad



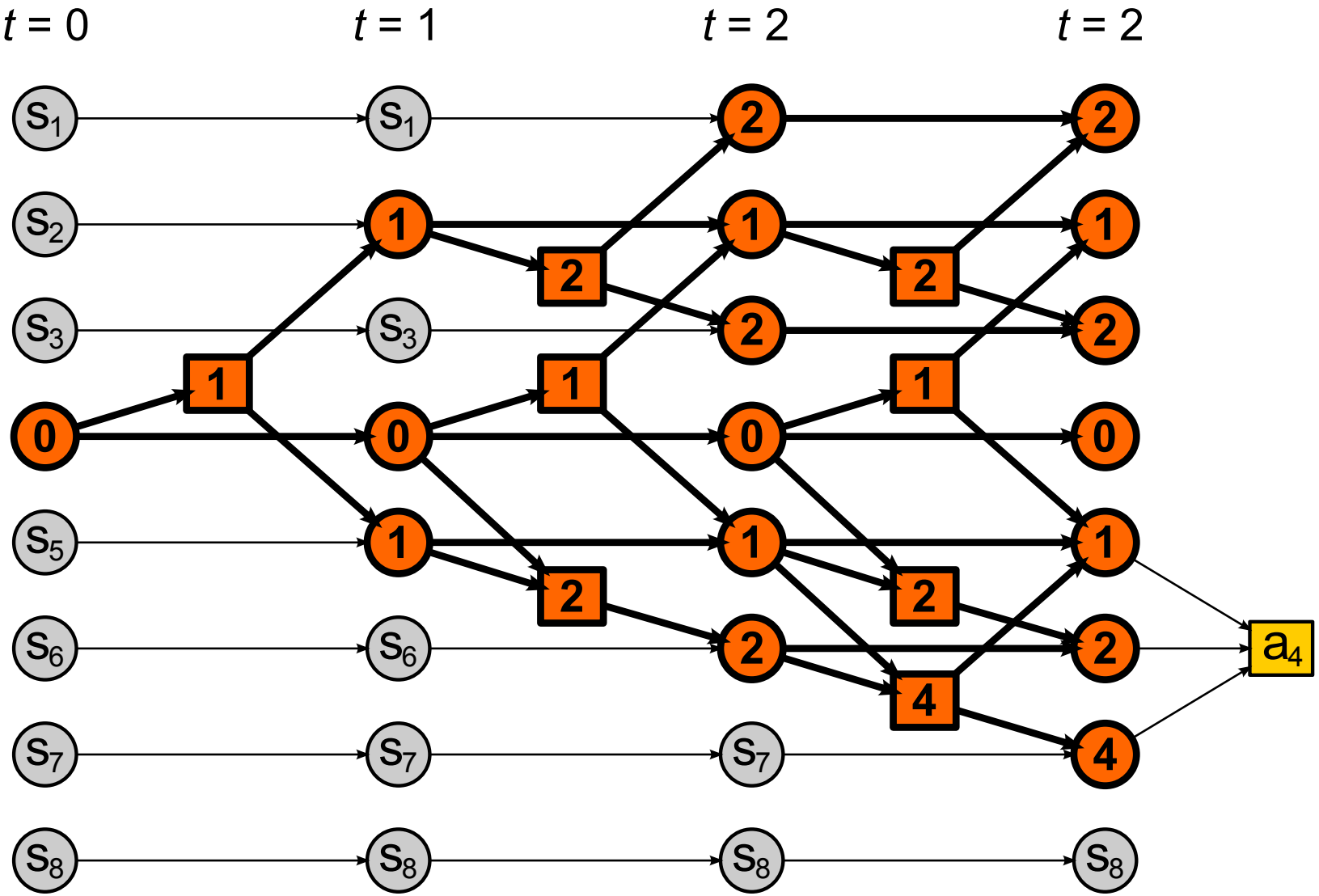
Heuristika h_{add}: Příklad



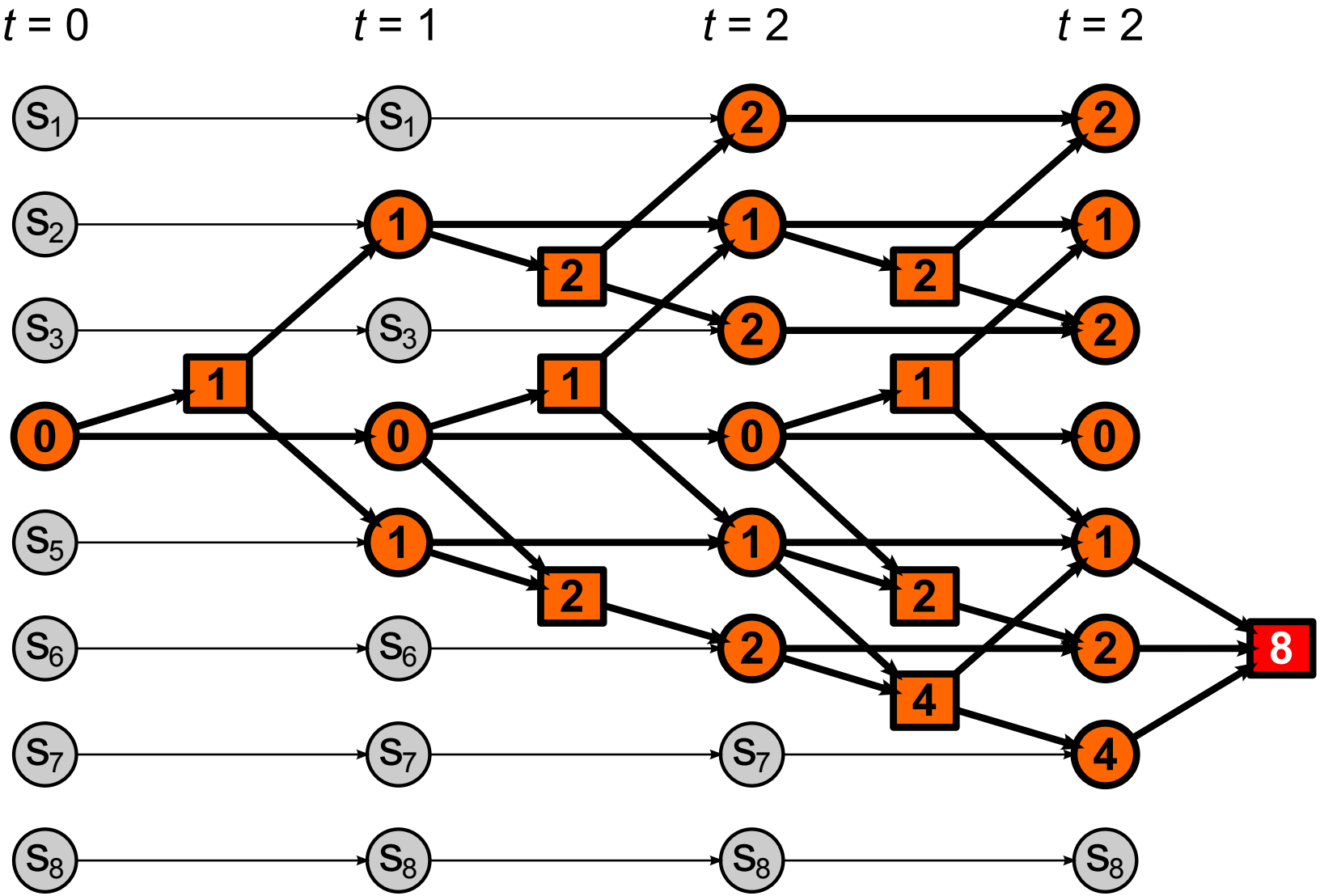
Heuristika h_{add}: Příklad



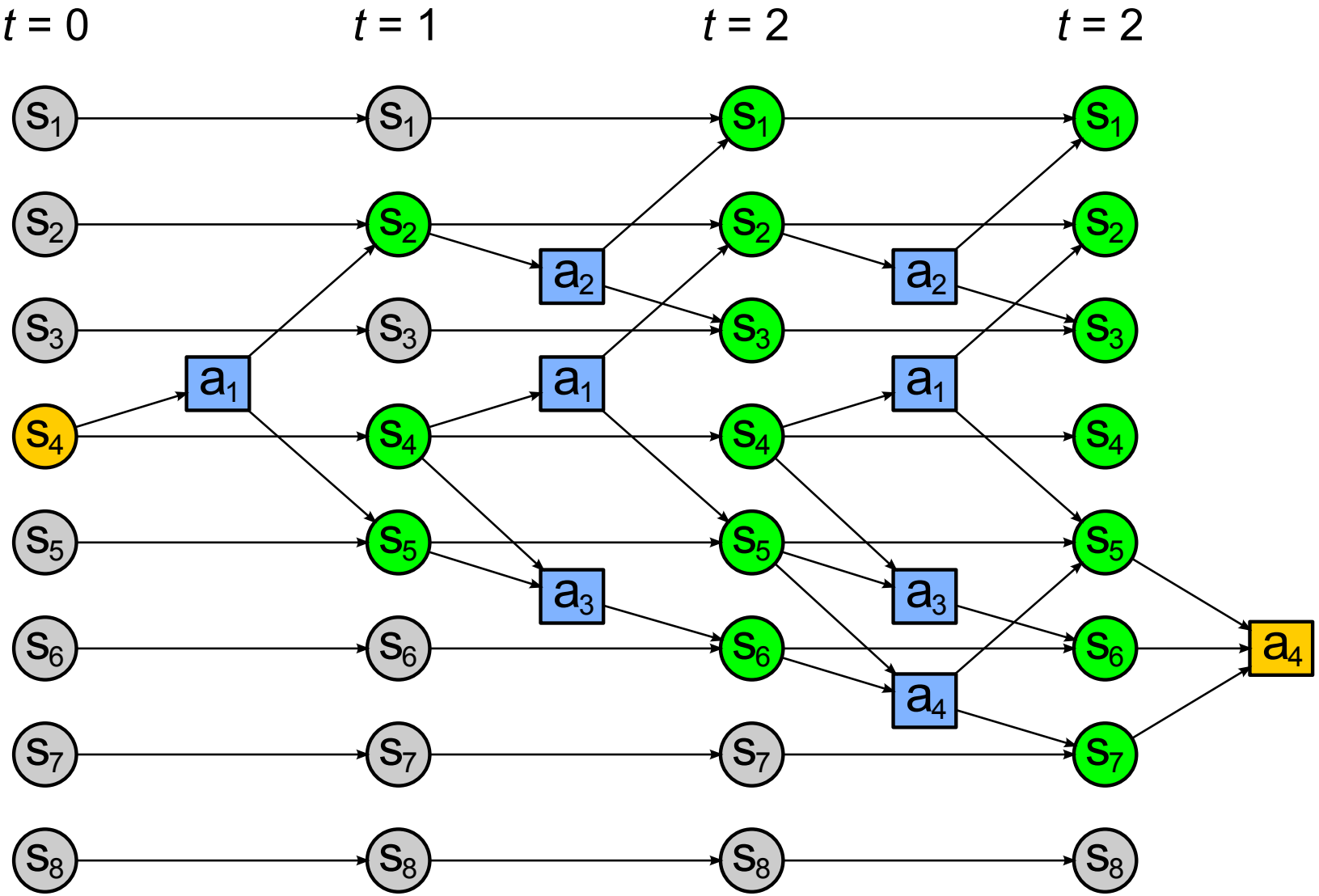
Heuristika h_{add}: Příklad



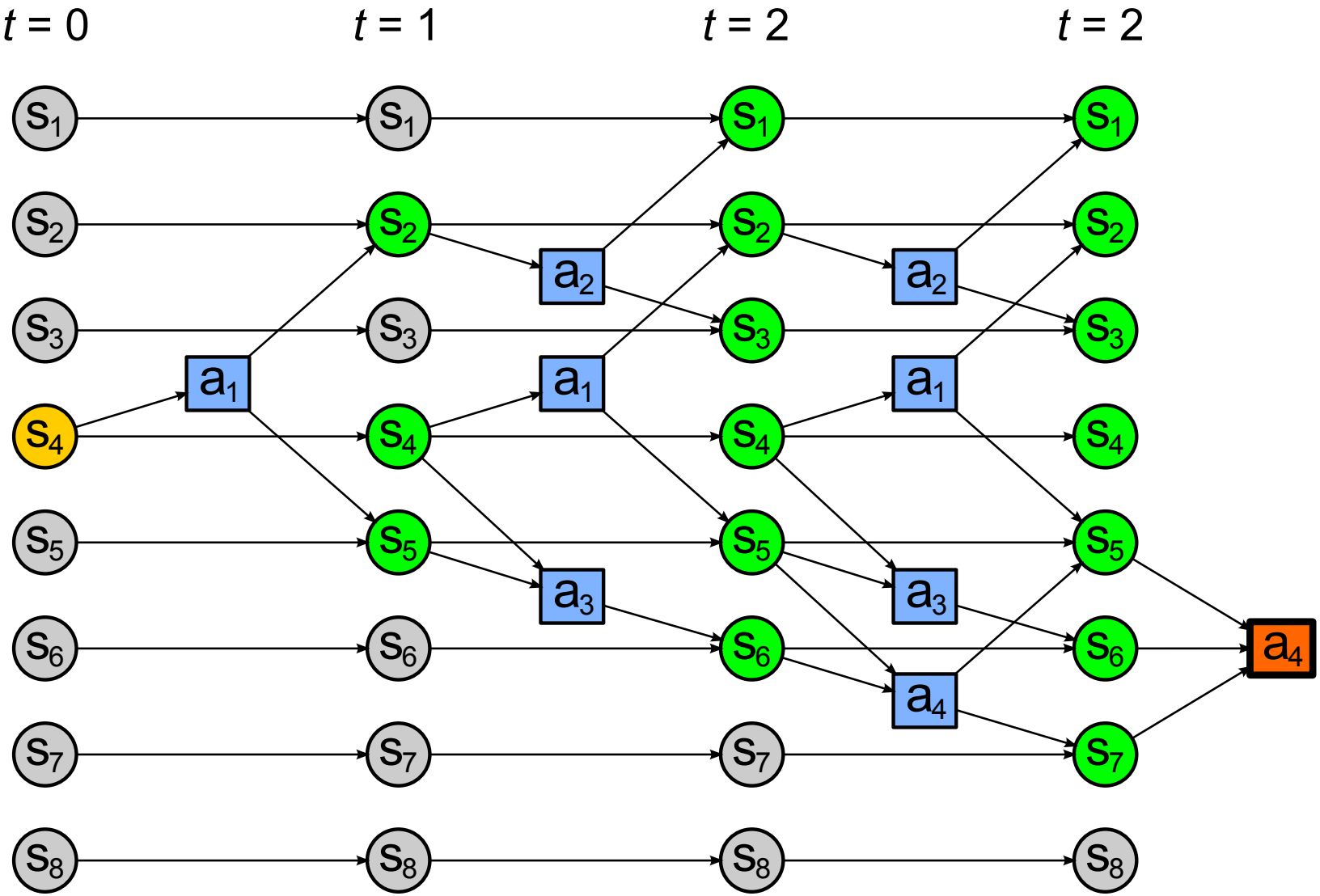
Heuristika h_{add}: Příklad



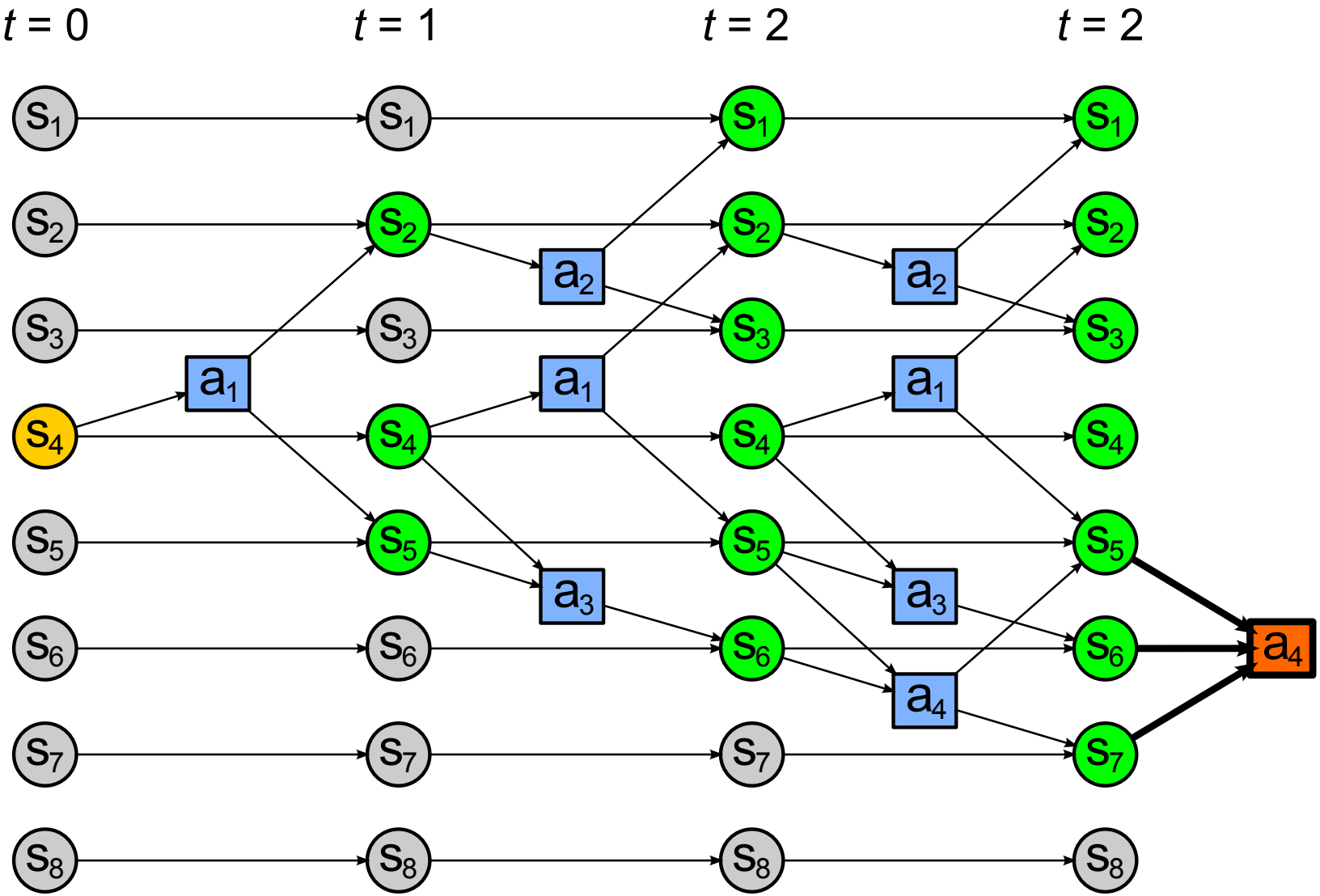
Heuristika h_{FF}: Příklad



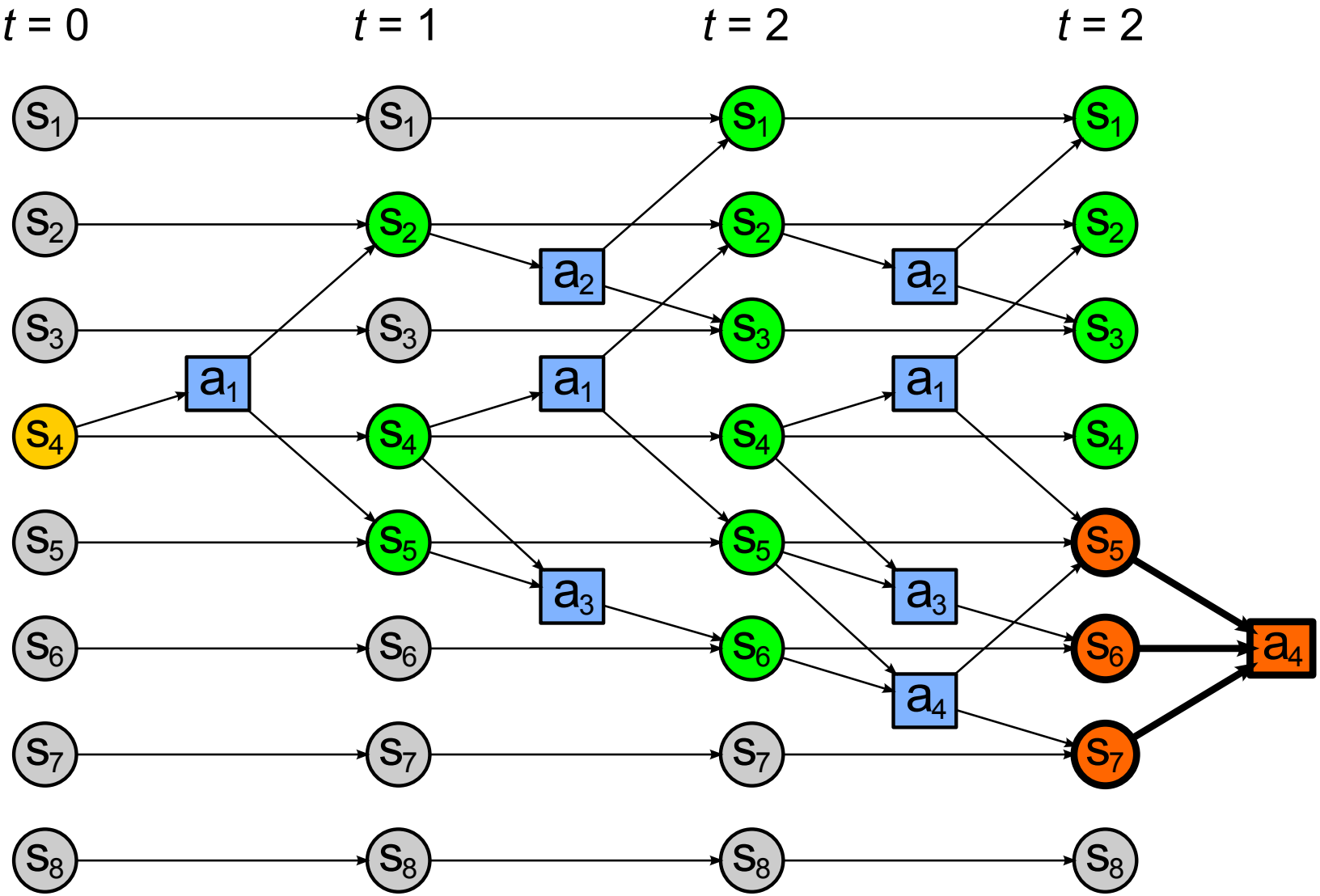
Heuristika h_{FF}: Příklad



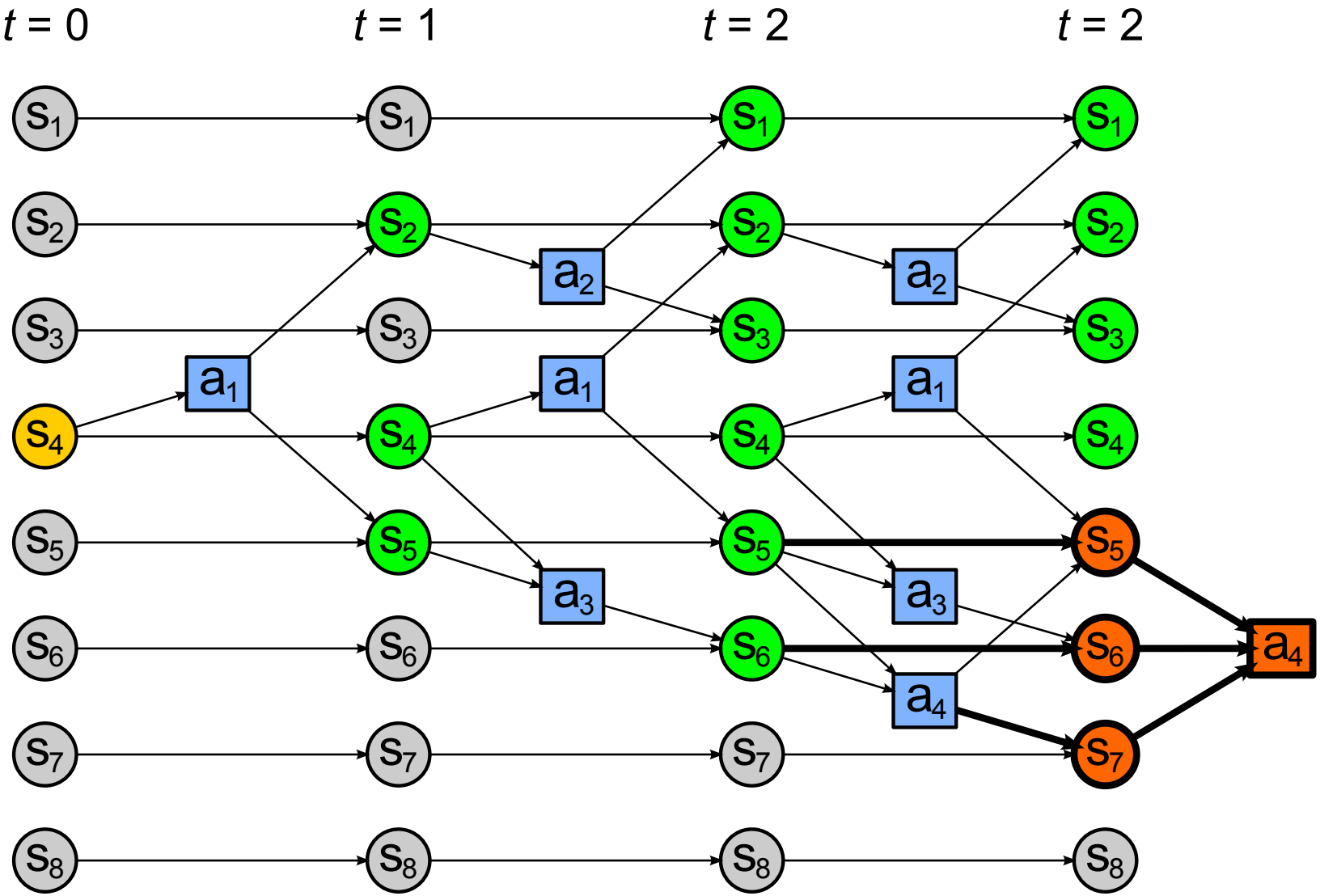
Heuristika h_{FF}: Příklad



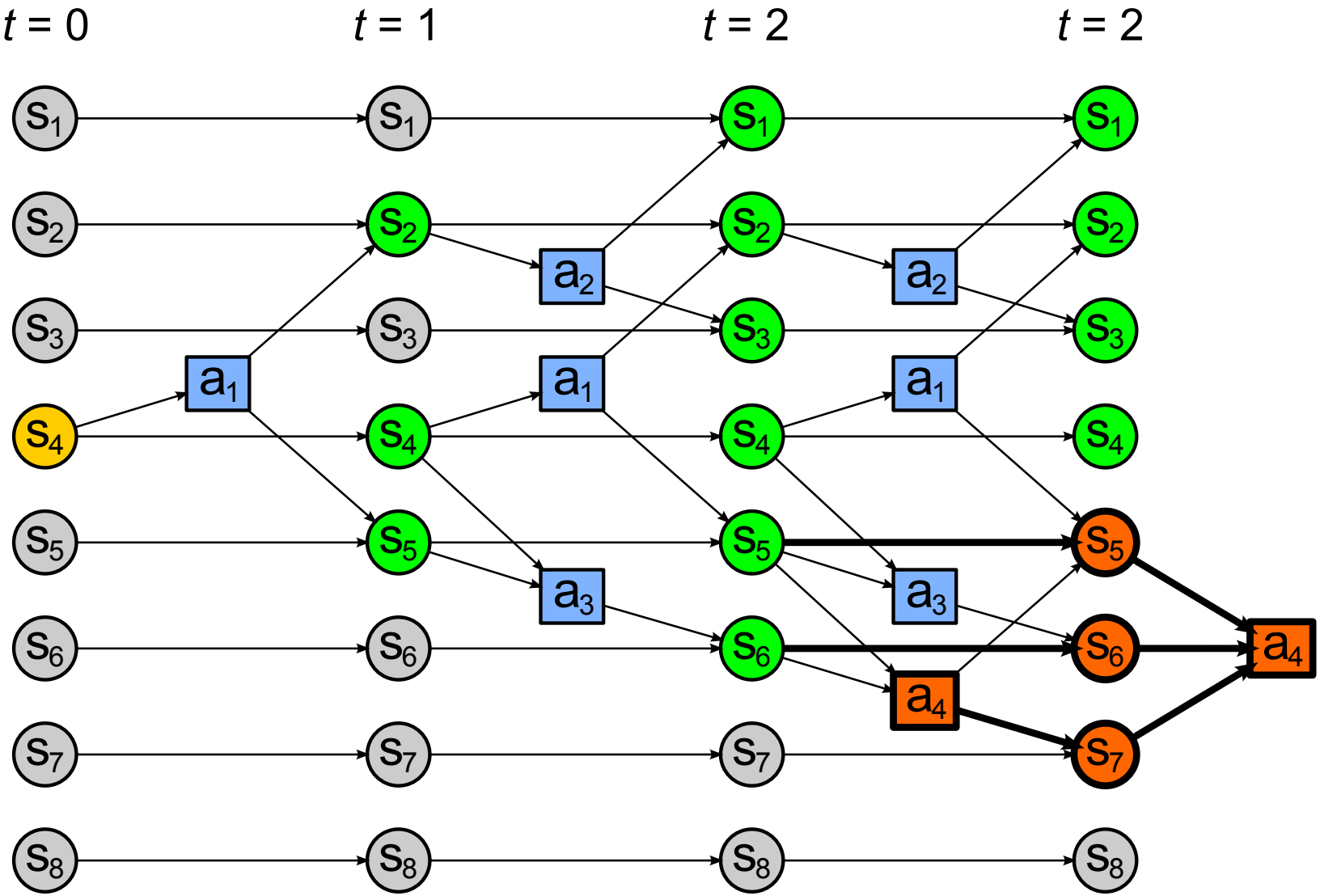
Heuristika h_{FF}: Příklad



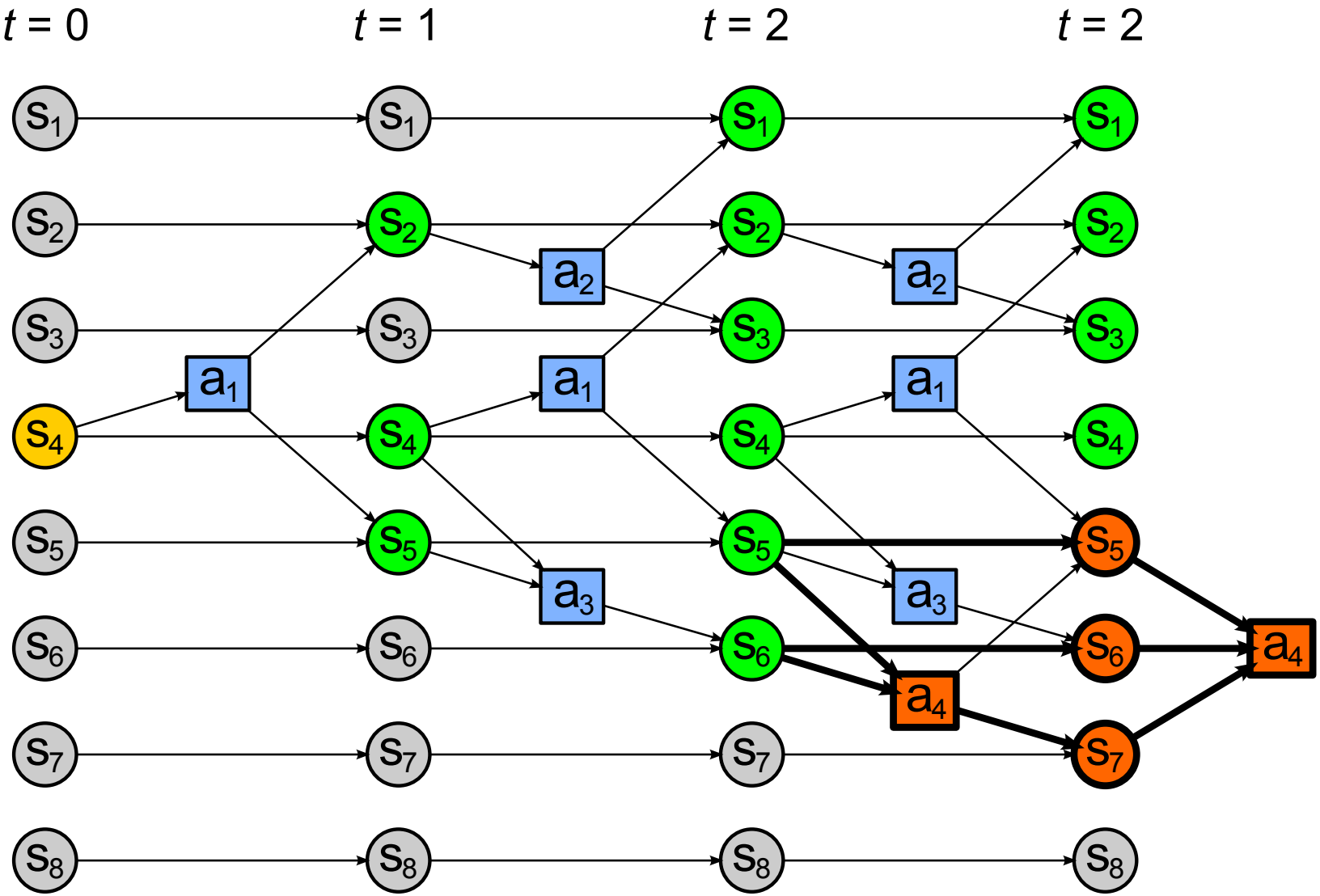
Heuristika h_{FF}: Příklad



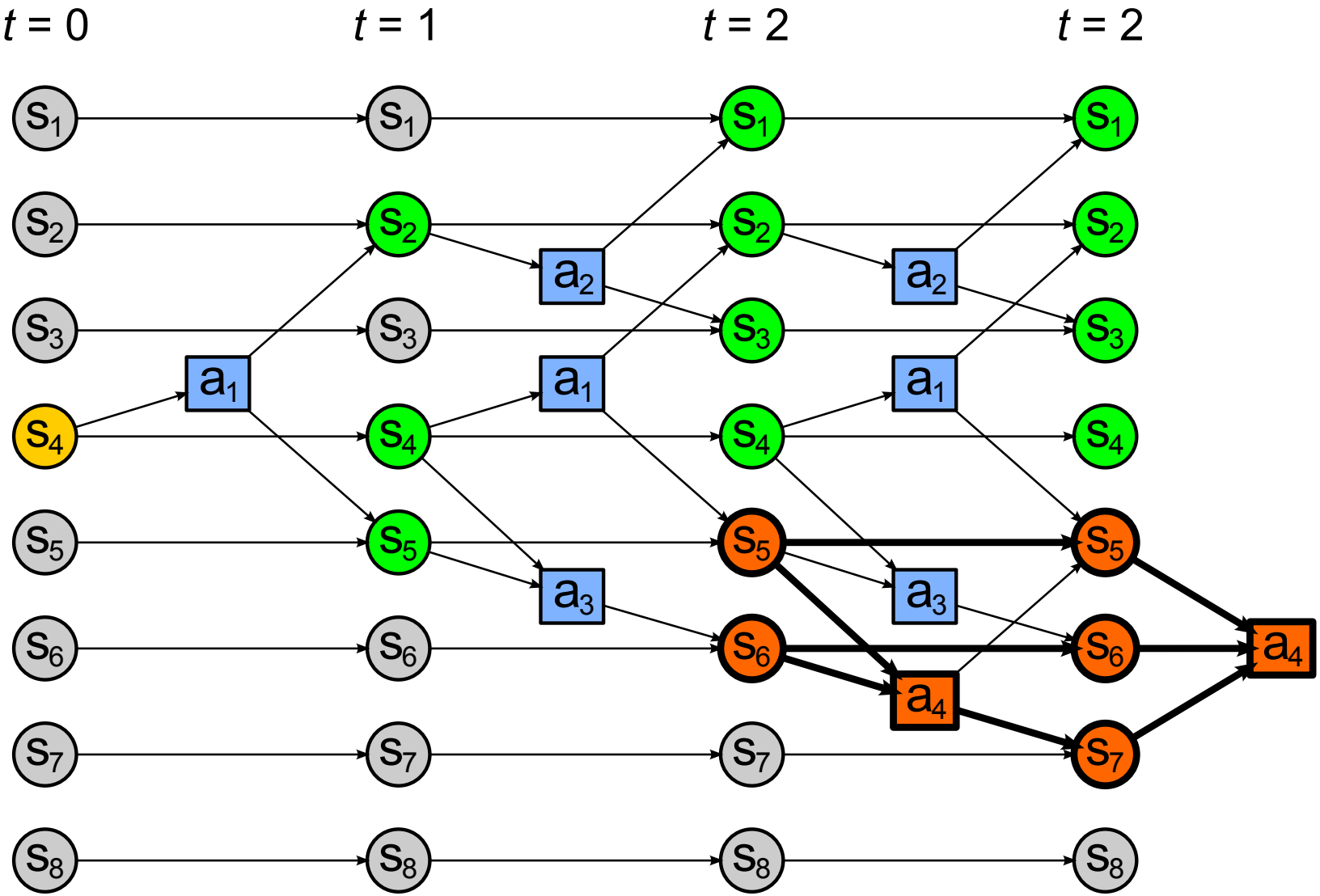
Heuristika h_{FF}: Příklad



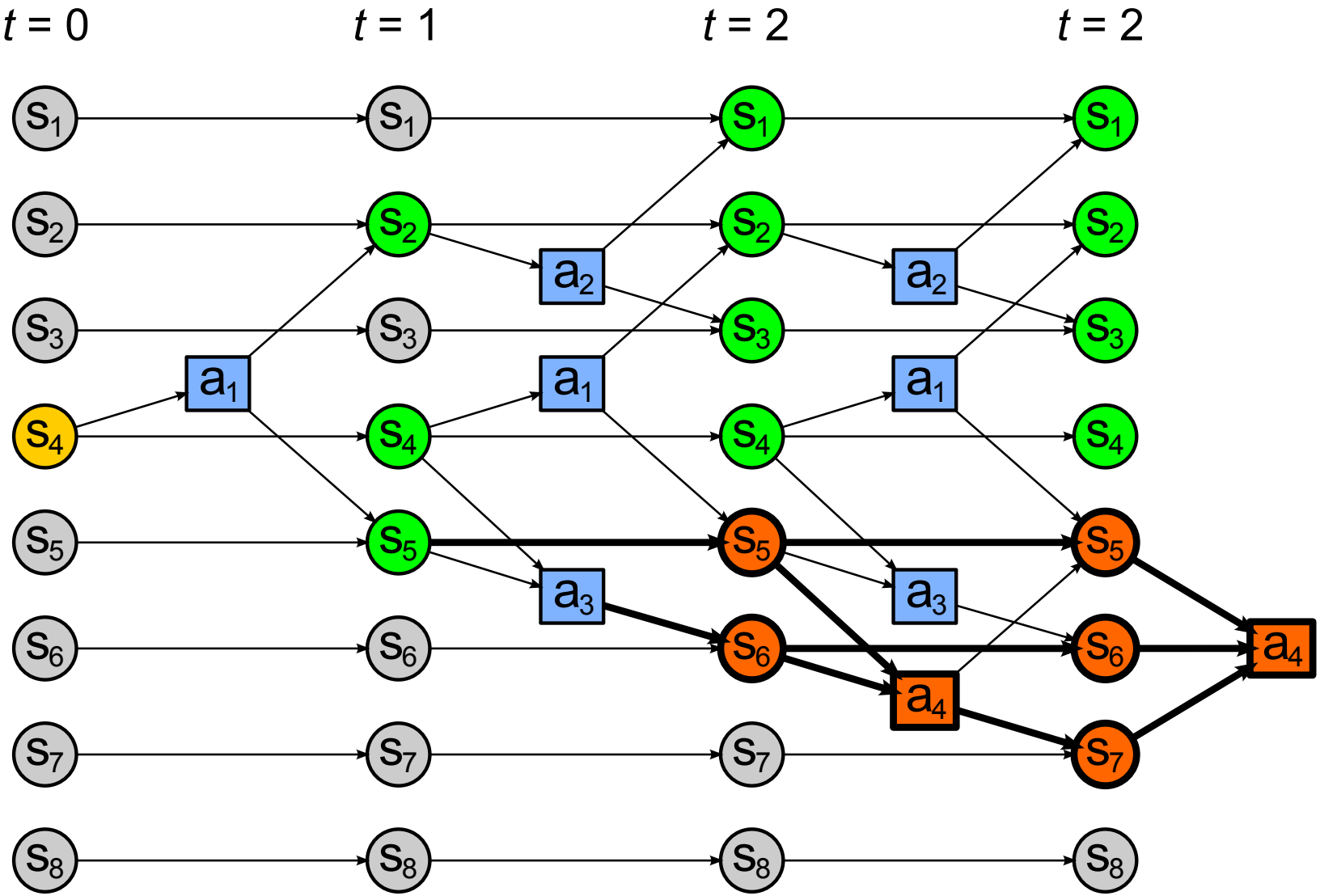
Heuristika h_{FF}: Příklad



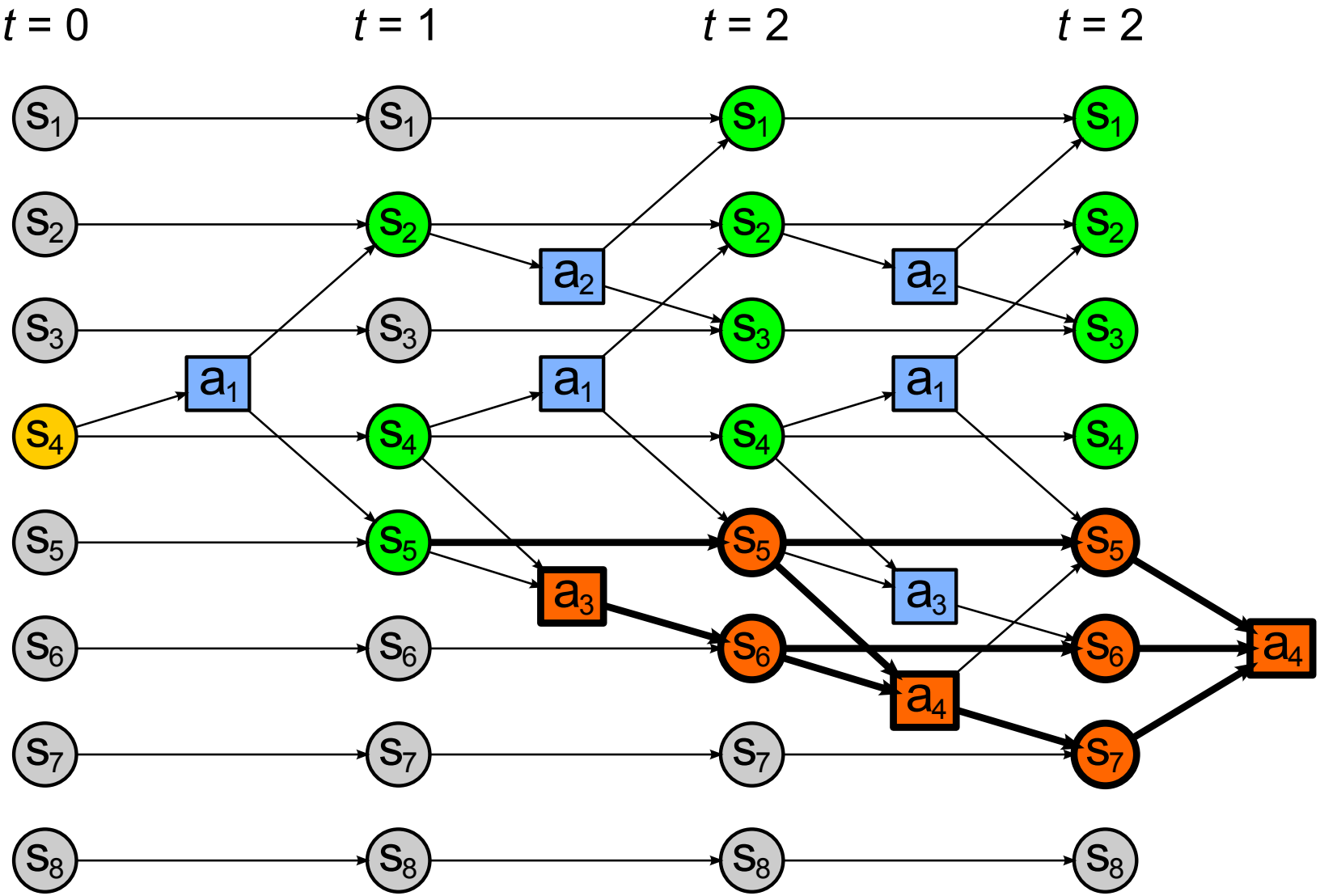
Heuristika h_{FF}: Příklad



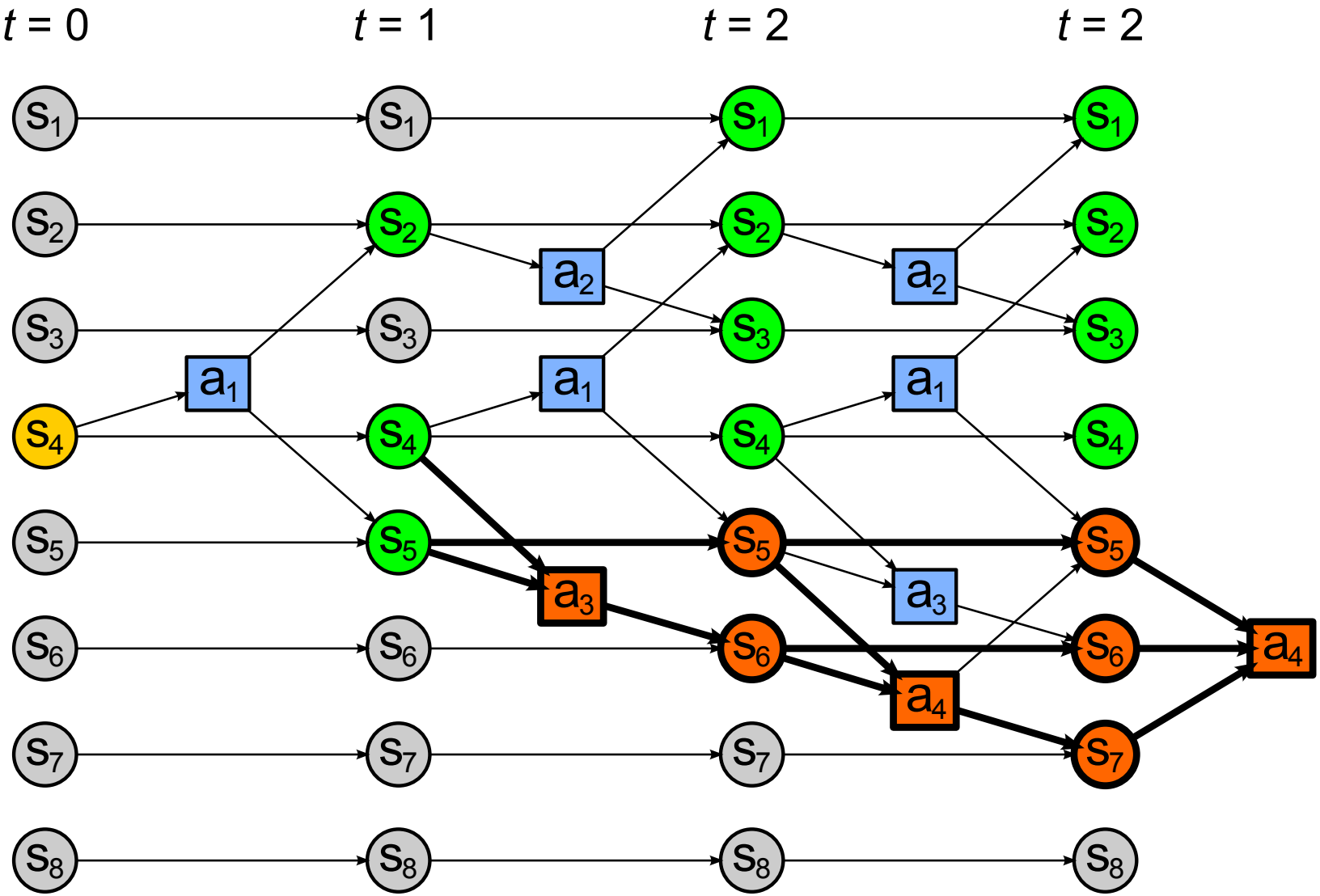
Heuristika h_{FF}: Příklad



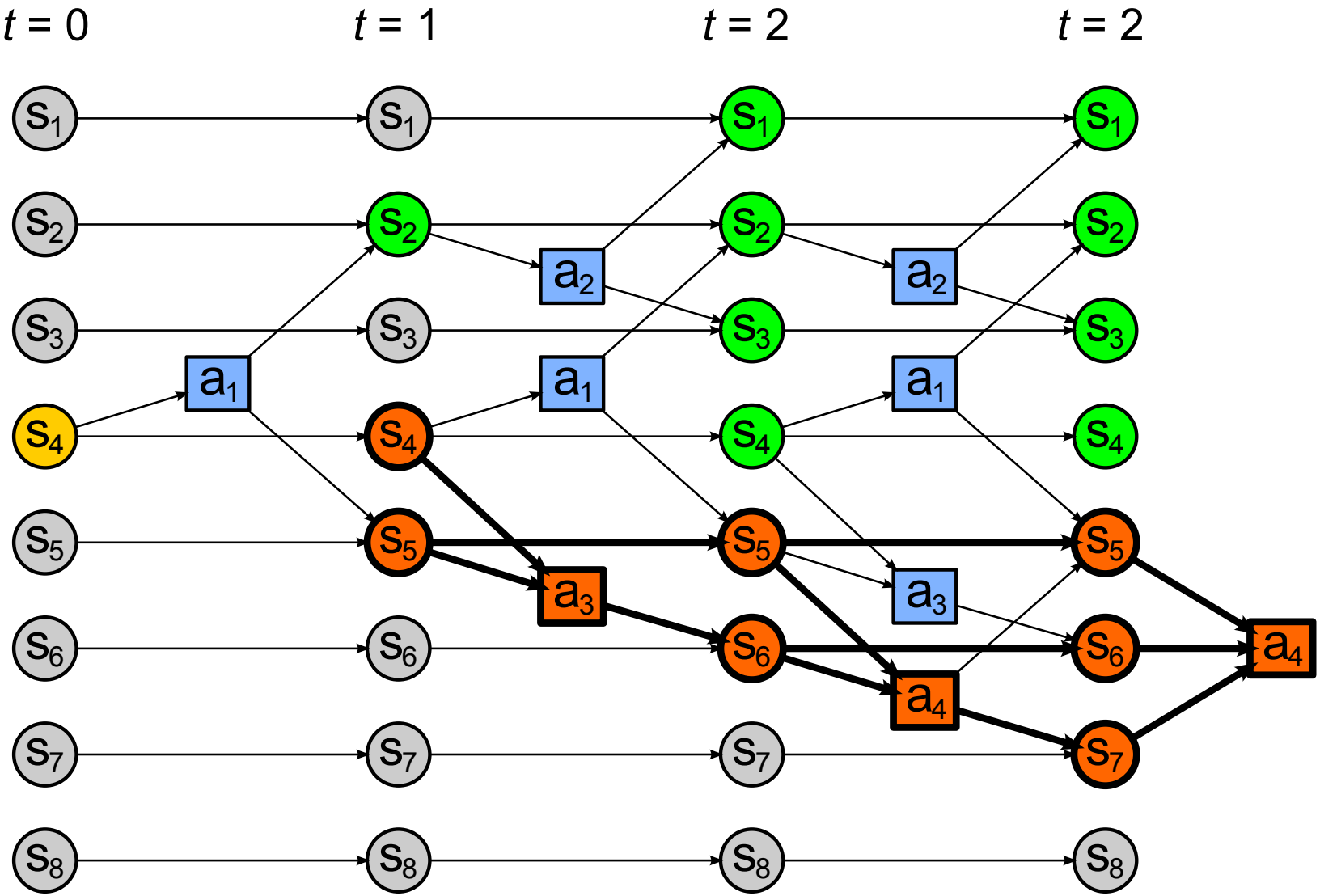
Heuristika h_{FF}: Příklad



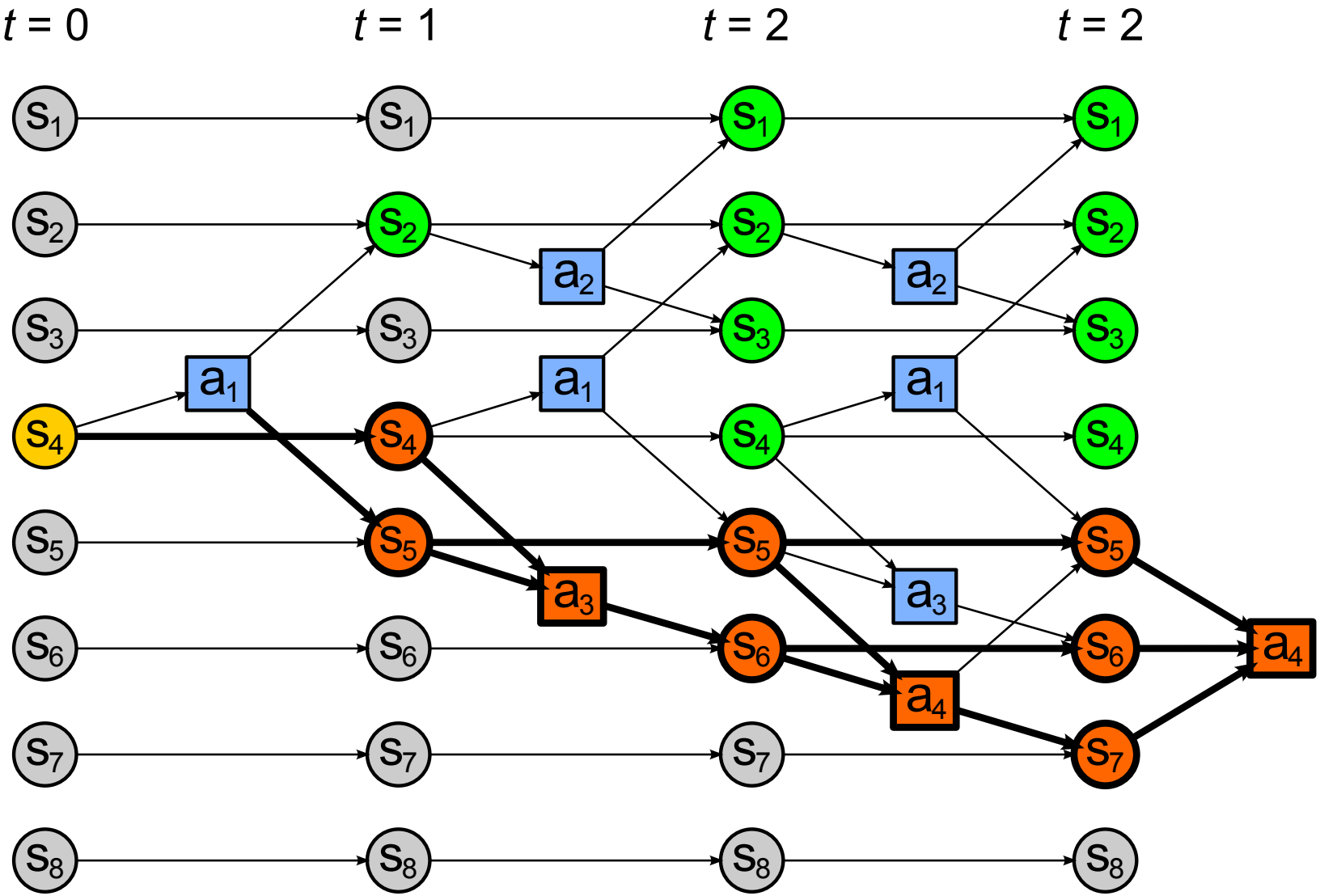
Heuristika h_{FF}: Příklad



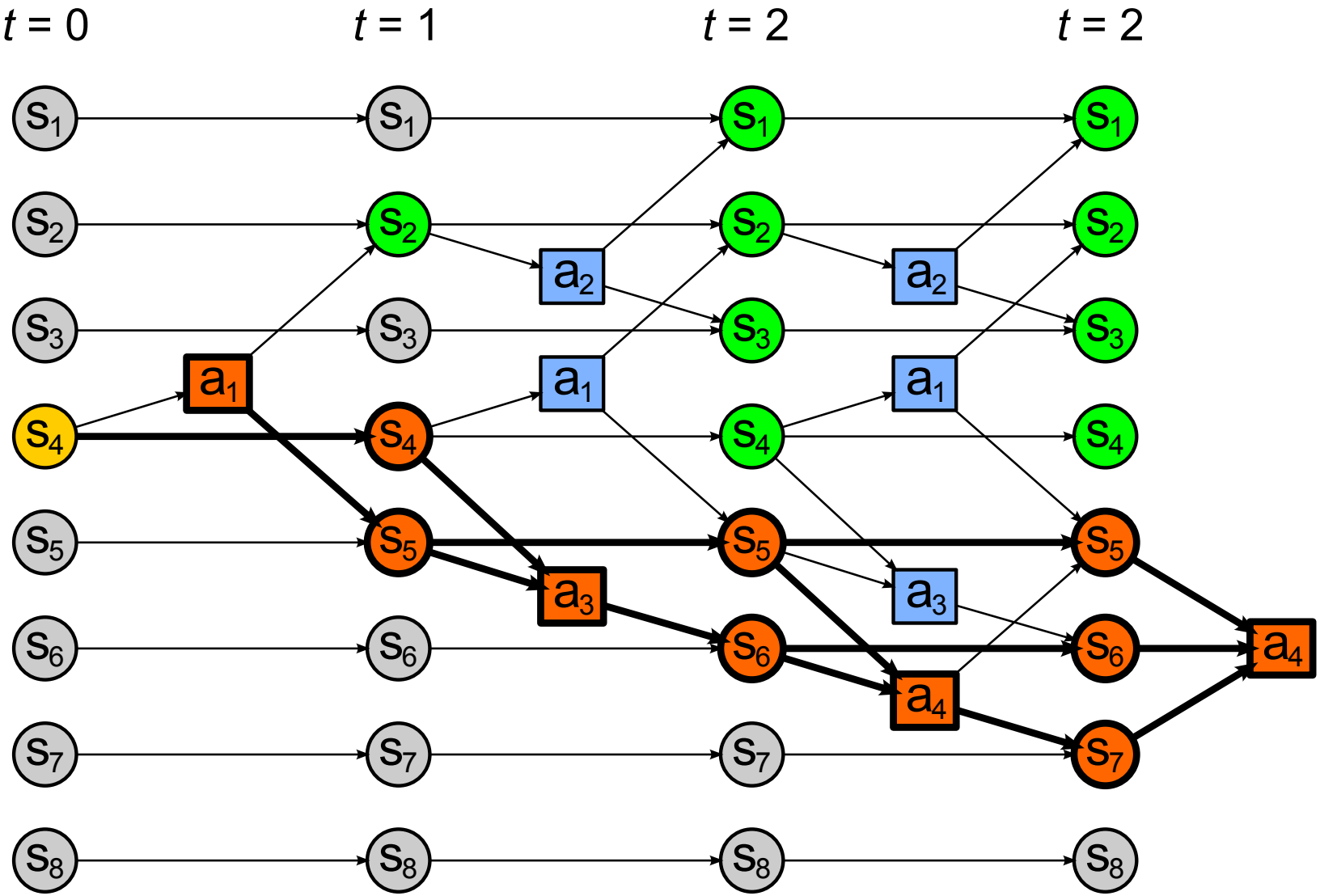
Heuristika h_{FF}: Příklad



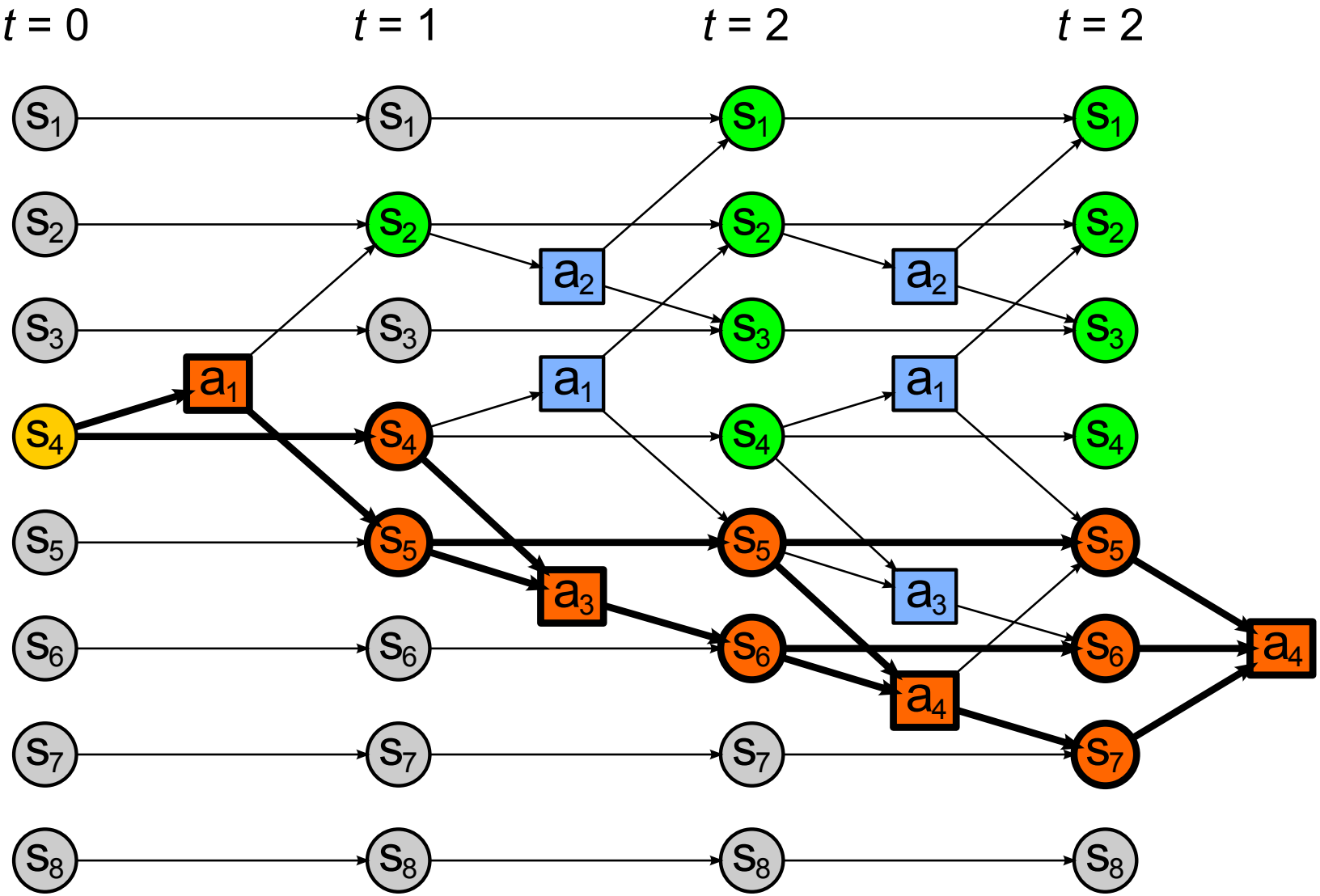
Heuristika h_{FF}: Příklad



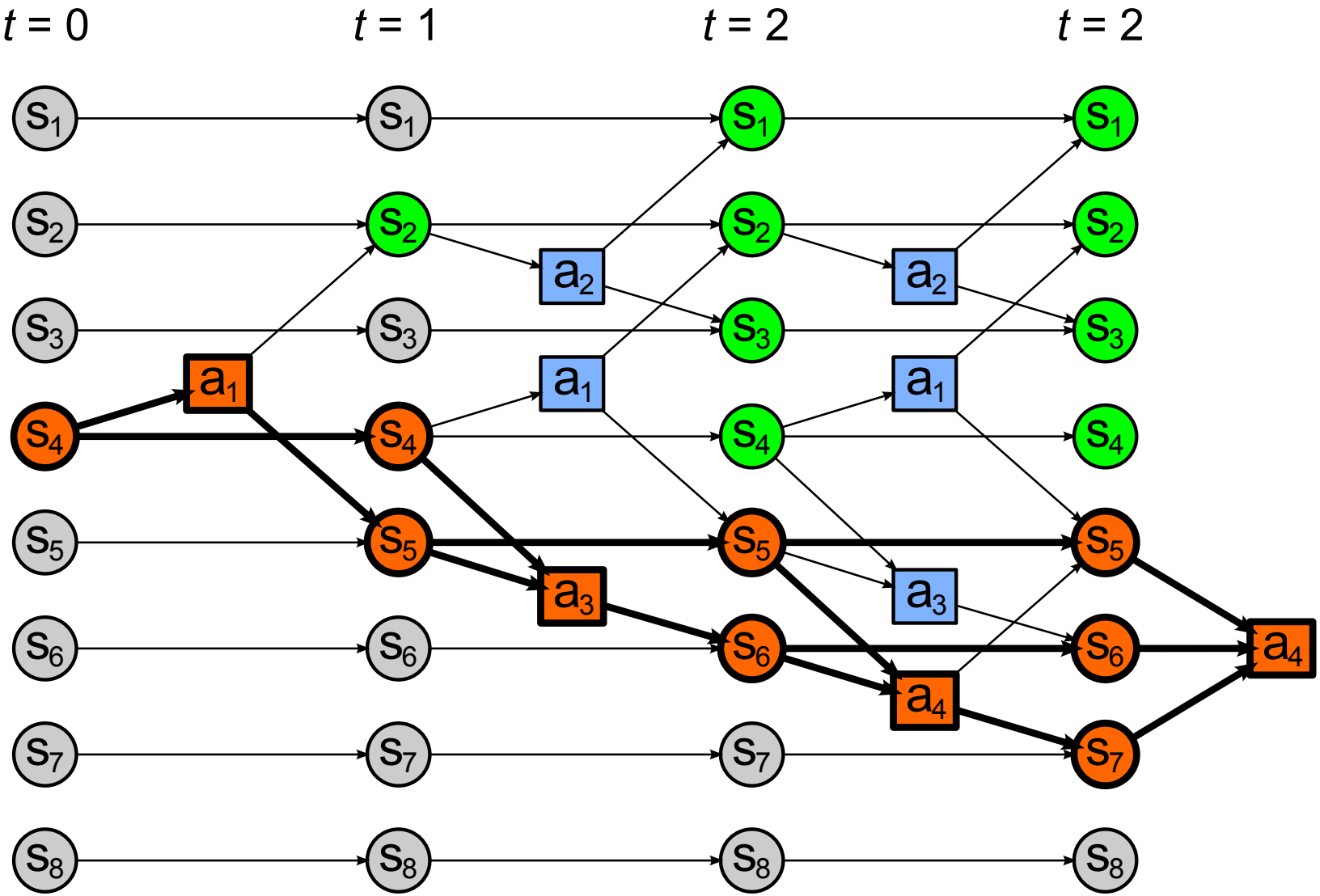
Heuristika h_{FF}: Příklad



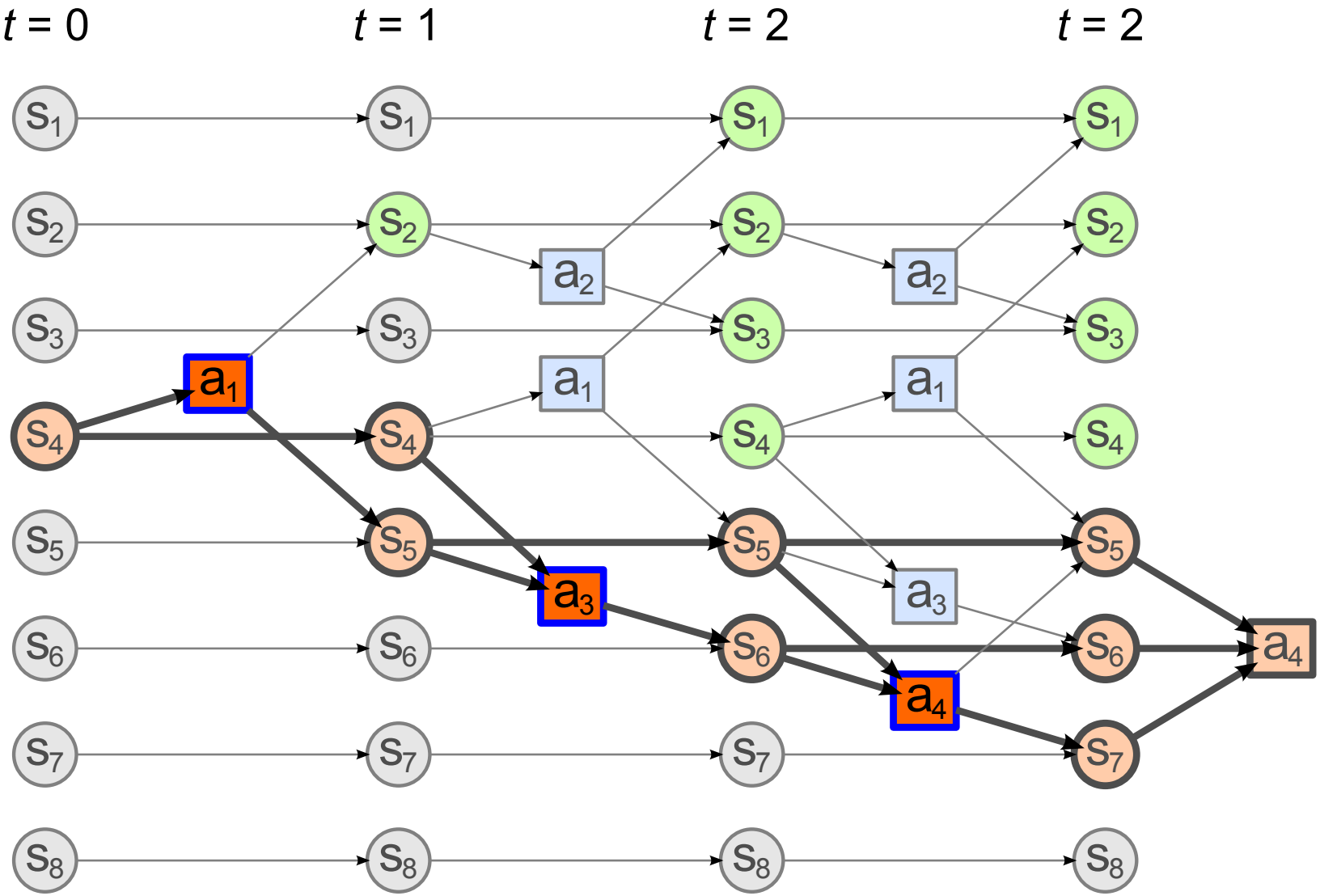
Heuristika h_{FF}: Příklad



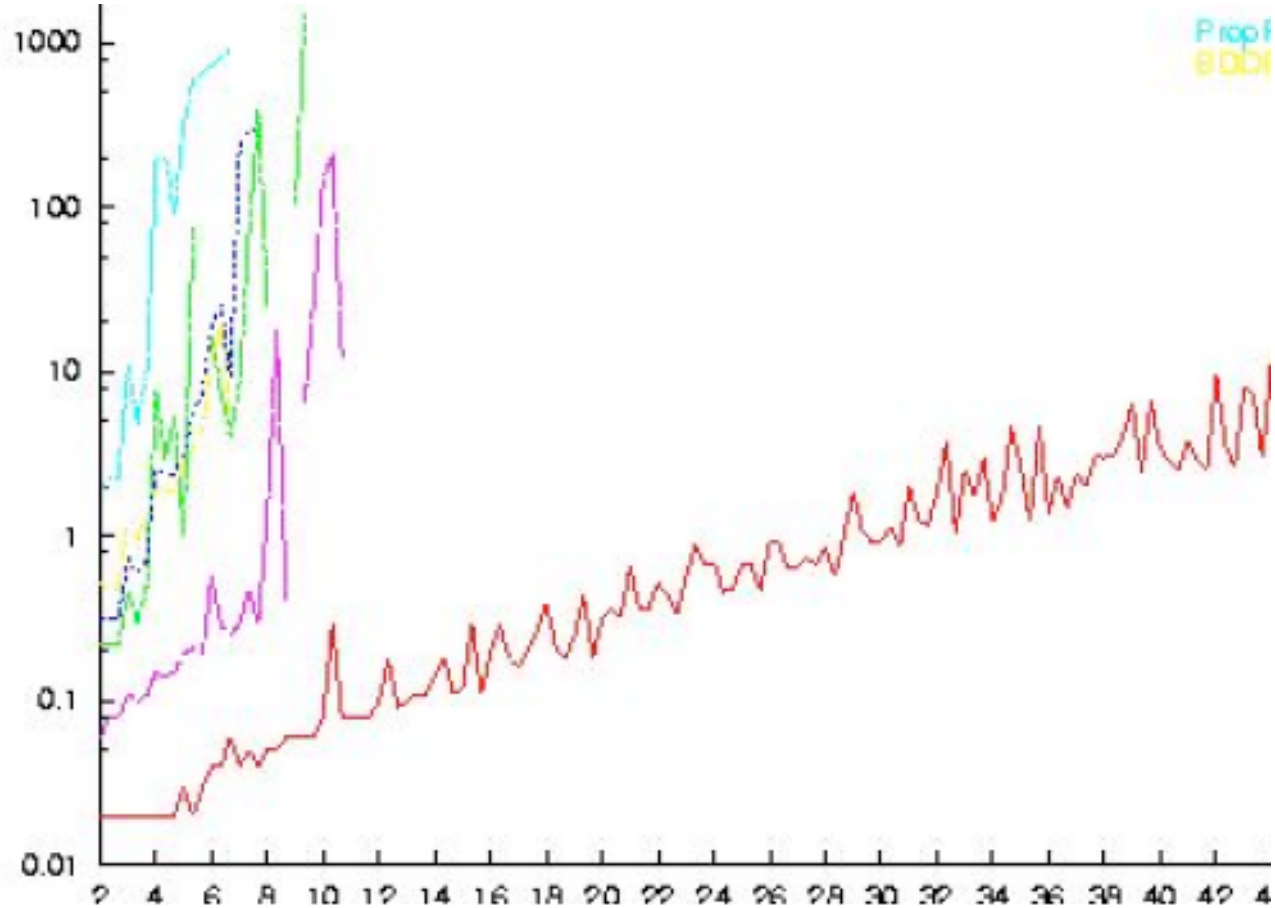
Heuristika h_{FF}: Příklad



Heuristika h_{FF}: Příklad



Heuristika h_{FF}: Porovnání se soudobými solvery



Abstrakce

- Alternativní přístup k tvorbě plánovacích heuristik, který ignoruje rozdíly mezi některými stavy,
- ke stavovému prostoru $\mathcal{S}$ vytvoříme **abstraktní stavový prostor** $\mathcal{S}'$ pomocí **abstrakčního zobrazení** α
- hlavní princip:
  - **ignorujeme některé atomy**,
  - různé stavy v původním prostoru tak mohou v prostoru abstraktním splynout,
  - v abstraktním prostoru následně hledáme optimální řešení
    - ★ složitost tohoto problému spočívá na míře abstrakce,
    - ★ výběr atomů k abstrakci: obrovská škála možností, je předmětem současného výzkumu
  - abstrakce se používá zásadně k tvorbě přípustných heuristik
    - ★ uplatnění v oblasti **optimálního plánování**

Princip abstrakce: Ilustrace