Faster Reachability Analysis for LR(1) Parsers

Frédéric Bour
Tarides & Inria
Paris, France

François Pottier
Inria
Paris, France

Plan

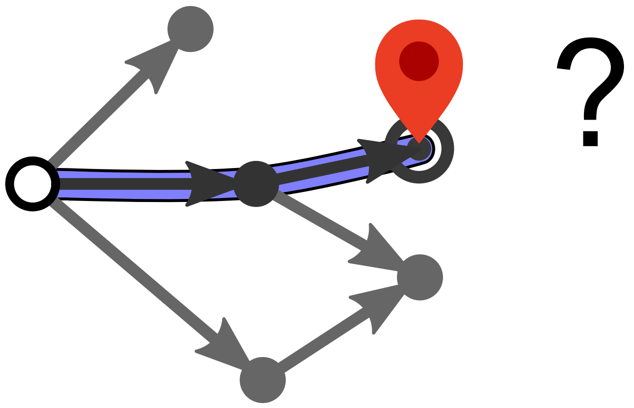
- The reachability problem for LR(1) automata
- State-of-the-art solution & performance comparison
- Main ideas of our contribution
- Conclusion

Reachability in LR(1) automata

What is the problem?

"Can the automaton reach a configuration (s,z) ?"

- s is the current state
- z is the first unconsumed symbol

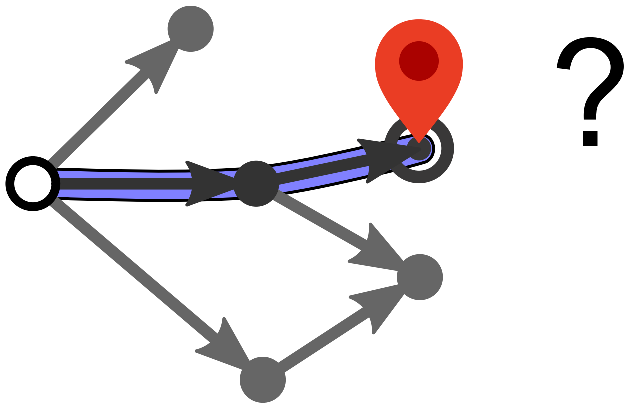


What is the problem?

“Can the automaton reach a configuration (s, z) ?”

- s is the current state
- z is the first unconsumed symbol

In practice, we also want a **minimal sentence** that reaches this configuration.



Why solve it?

Why solve it?

- Test case generation

Why solve it?

- Test case generation
  - **Negative test cases** (our main focus)
    Enumerate sentences that cause errors in all states that can fail
    (Jeffery 2003, Pottier 2016)

Why solve it?

- Test case generation
  - **Negative test cases** (our main focus)
    Enumerate sentences that cause errors in all states that can fail
    (Jeffery 2003, Pottier 2016)

Assistance to write error message:

```
translation_unit_file: INT PRE_NAME VAR_NAME EQ XOR_ASSIGN
## Ends in an error in state: 561.

Ill-formed init declarator.
At this point, an initializer is expected.
```

Why solve it?

- Test case generation
  - **Negative test cases** (our main focus)

    Enumerate sentences that cause errors in all states that can fail
    (Jeffery 2003, Pottier 2016)
  - Positive test cases

    Cover all reductions for regression testing,
    check compatibility between different grammar versions,
    ...

Why solve it?

- Test case generation
  - **Negative test cases** (our main focus)

    Enumerate sentences that cause errors in all states that can fail
    (Jeffery 2003, Pottier 2016)
  - Positive test cases

    Cover all reductions for regression testing,
    check compatibility between different grammar versions,
    ...
- Syntactic completion, syntactic error recovery, ...

State-of-the-art solution & performance comparison

Pottier’s algorithm (2016)

Pottier's algorithm (2016)

- Implemented in the Menhir parser generator

Pottier's algorithm (2016)

- Implemented in the Menhir parser generator
- Applied to CompCert to obtain high-quality error messages

Pottier's algorithm (2016)

- Implemented in the Menhir parser generator
- Applied to CompCert to obtain high-quality error messages

But it does not scale well!

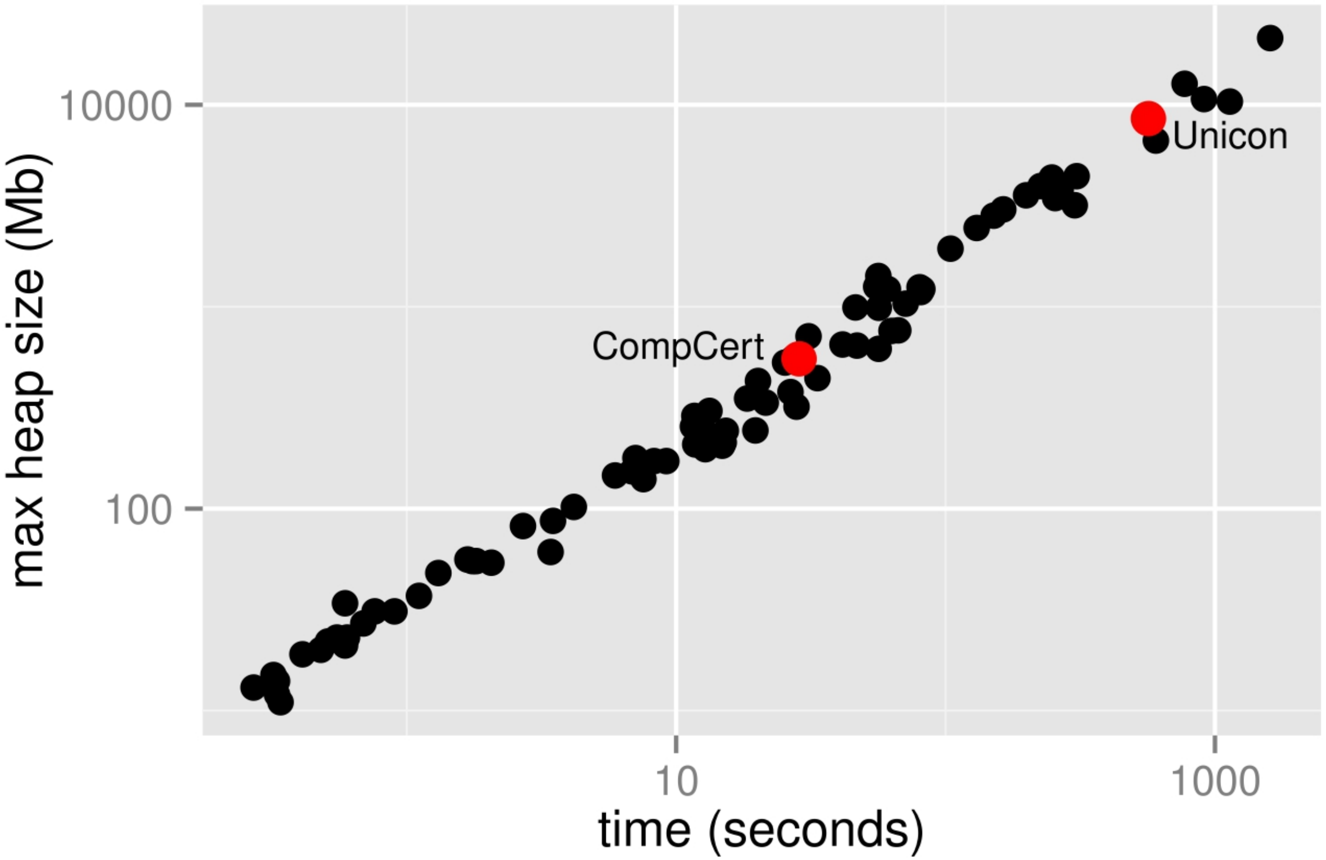
Pottier’s algorithm (2016)

A few data points:

- CompCert (C): 25s and 529MB
- Unicon: 566s and 8.5GB

Problems:

- Too slow for interactive use
- Painful for grammar maintainers



Pottier's algorithm (2016)

The algorithm works in two steps:

Pottier’s algorithm (2016)

The algorithm works in two steps:

1. For **each transition**, find the **shortest input** that allows taking it (while satisfying constraints on lookahead tokens)

Pottier’s algorithm (2016)

The algorithm works in two steps:

1. For **each transition**, find the **shortest input** that allows taking it (while satisfying constraints on lookahead tokens)
2. Generate minimal sentences by taking consecutive transitions

Pottier's algorithm (2016)

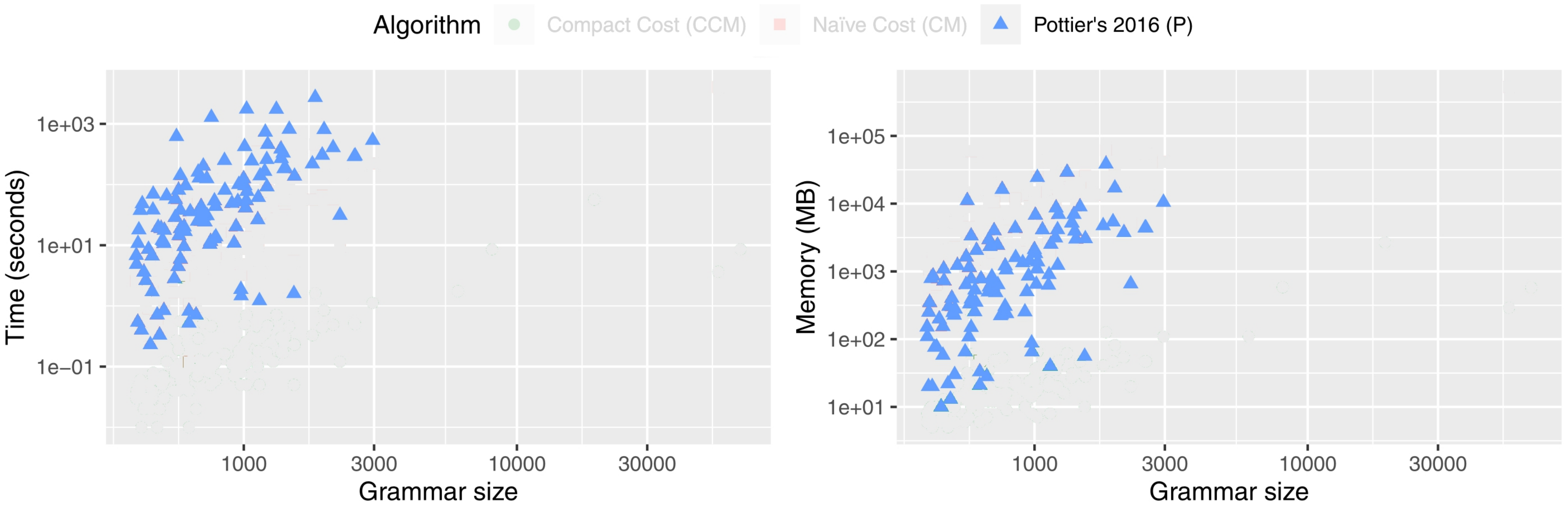
The algorithm works in two steps:

1. For **each transition**, find the **shortest input** that allows taking it (while satisfying constraints on lookahead tokens)
2. Generate minimal sentences by taking consecutive transitions

The bottleneck by far is step 1.

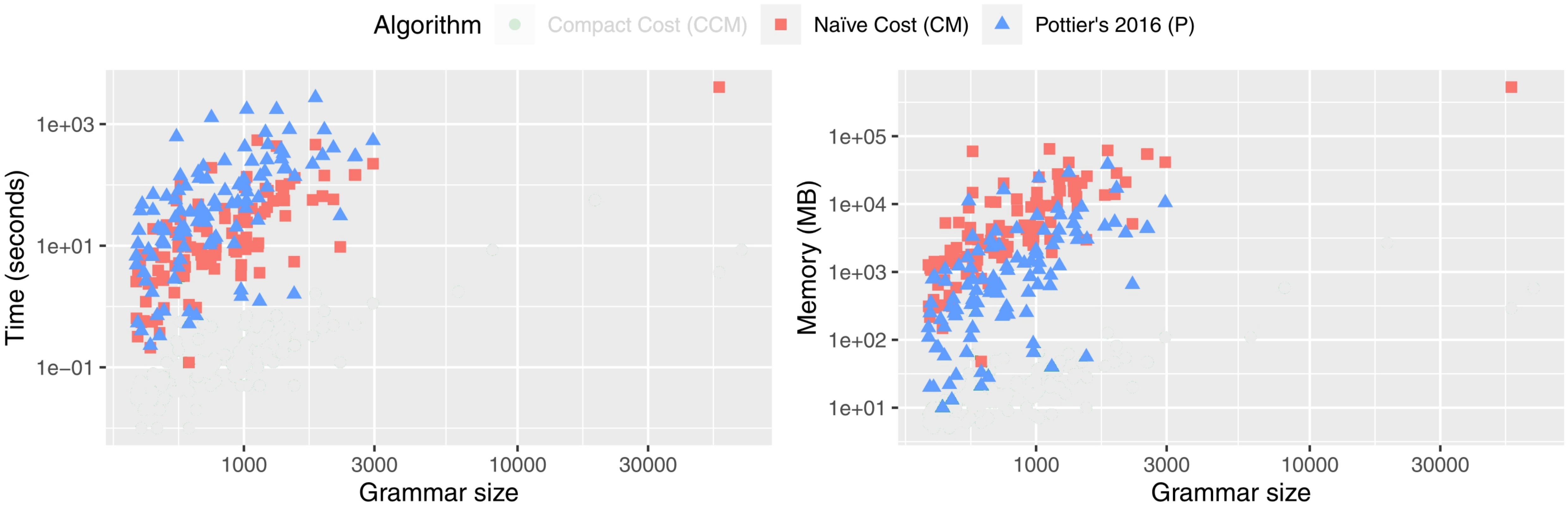
We propose a new algorithm to solve it.

Our contribution: speeding up the analysis!



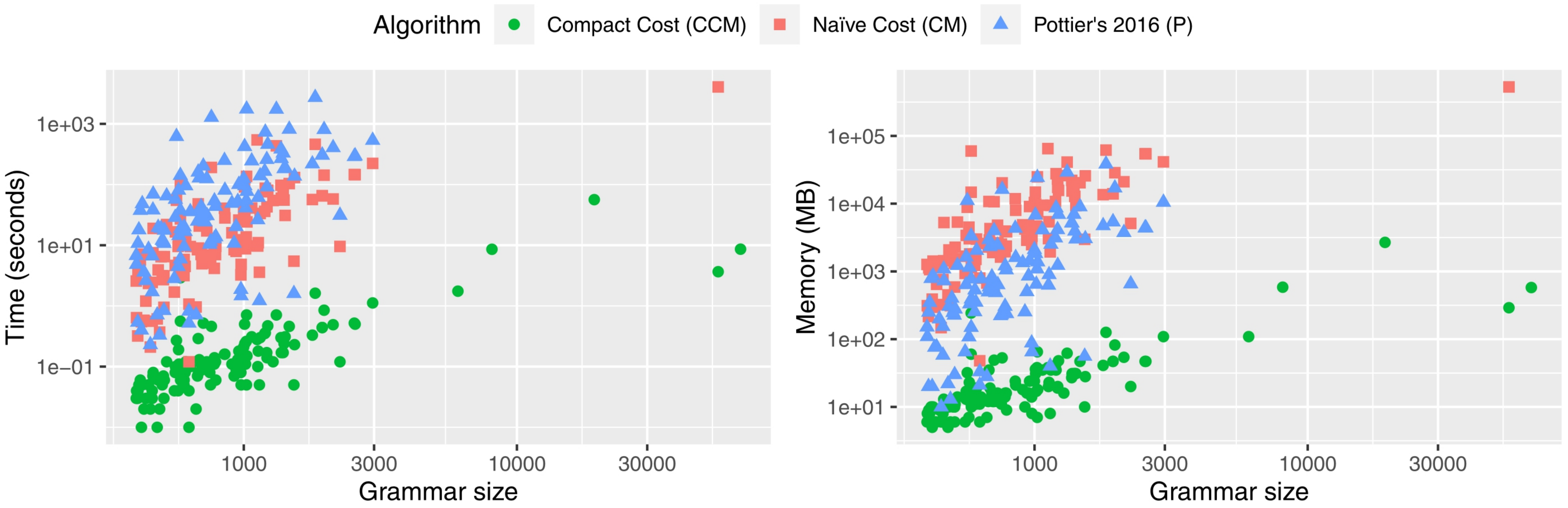
Original algorithm.

Our contribution: speeding up the analysis!



First step: a "naïve" matrix-based formulation (faster! but memory hungry)

Our contribution: speeding up the analysis!



Second step: compact matrices, **two to three orders of magnitude better**, in time and space.

Our contribution: speeding up the analysis!

Updated data points:

- CompCert (C): **0.10s and 12MB** (was 25s and 529MB)
- Unicon: **0.28s and 32MB** (was 566s and 8.5GB)

Our contribution: speeding up the analysis!

Updated data points:

- CompCert (C): **0.10s and 12MB** (was 25s and 529MB)
- Unicon: **0.28s and 32MB** (was 566s and 8.5GB)

Can still take some time: a “rich” C++ grammar that takes 56s and 2.7GB.

(grammar from “Diff/TS: A tool for fine-grained structural change analysis” by Hashimoto and Mori)

Idea #1: costs with matrices

An example grammar

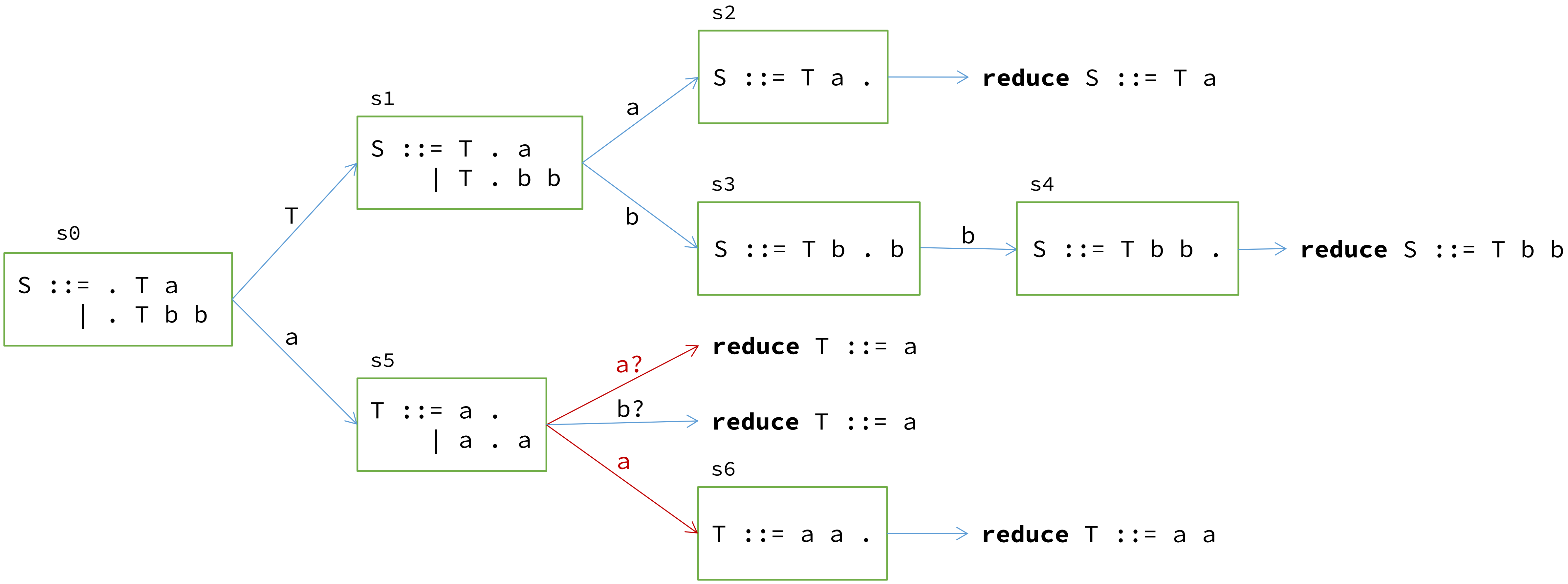
Let’s consider this LR(1) grammar:

```
S ::= T a
    | T b b

T ::= a
    | a a
```

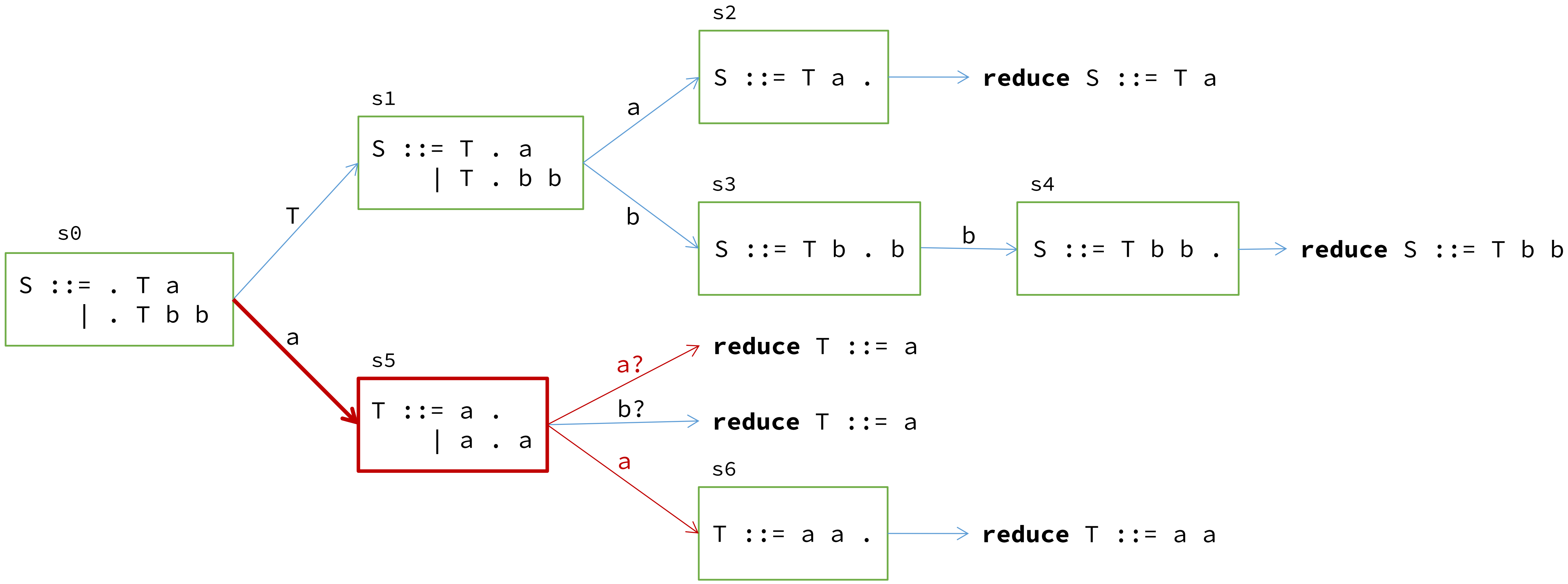
The automaton

It turns into the following LR(1) automaton, with one `SHIFT/REDUCE` conflict



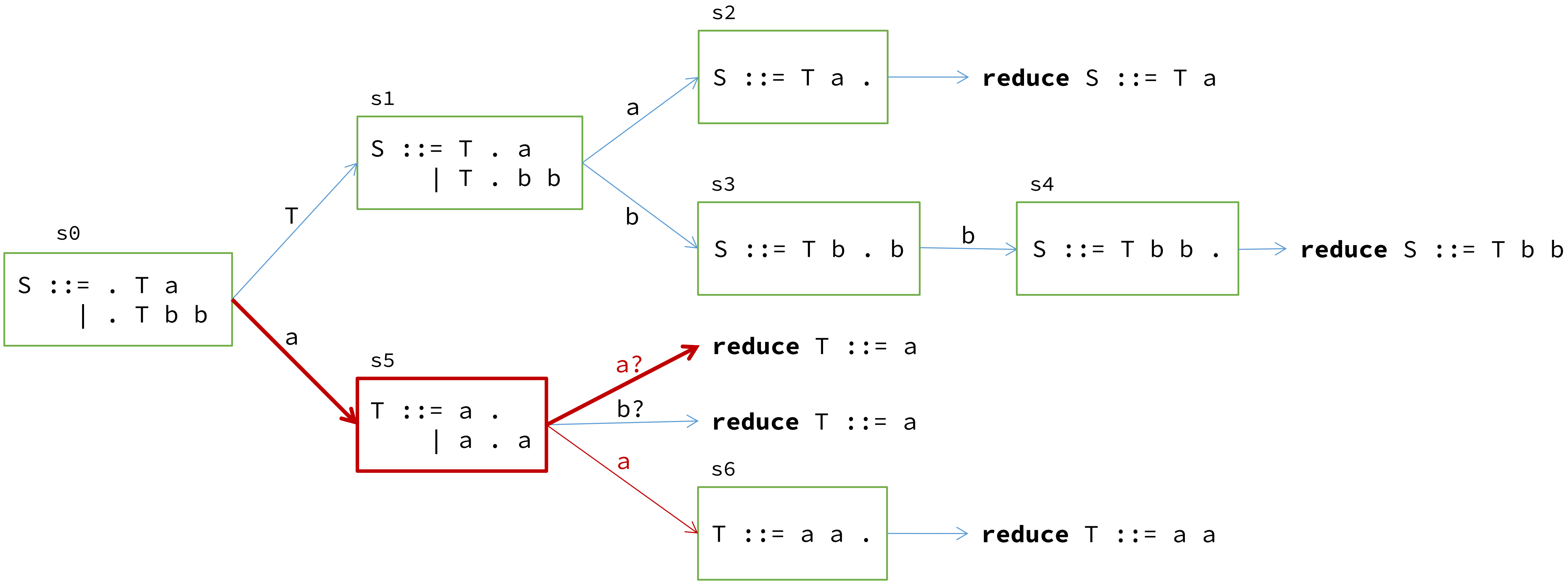
The automaton

It turns into the following LR(1) automaton, with one `SHIFT/REDUCE` conflict



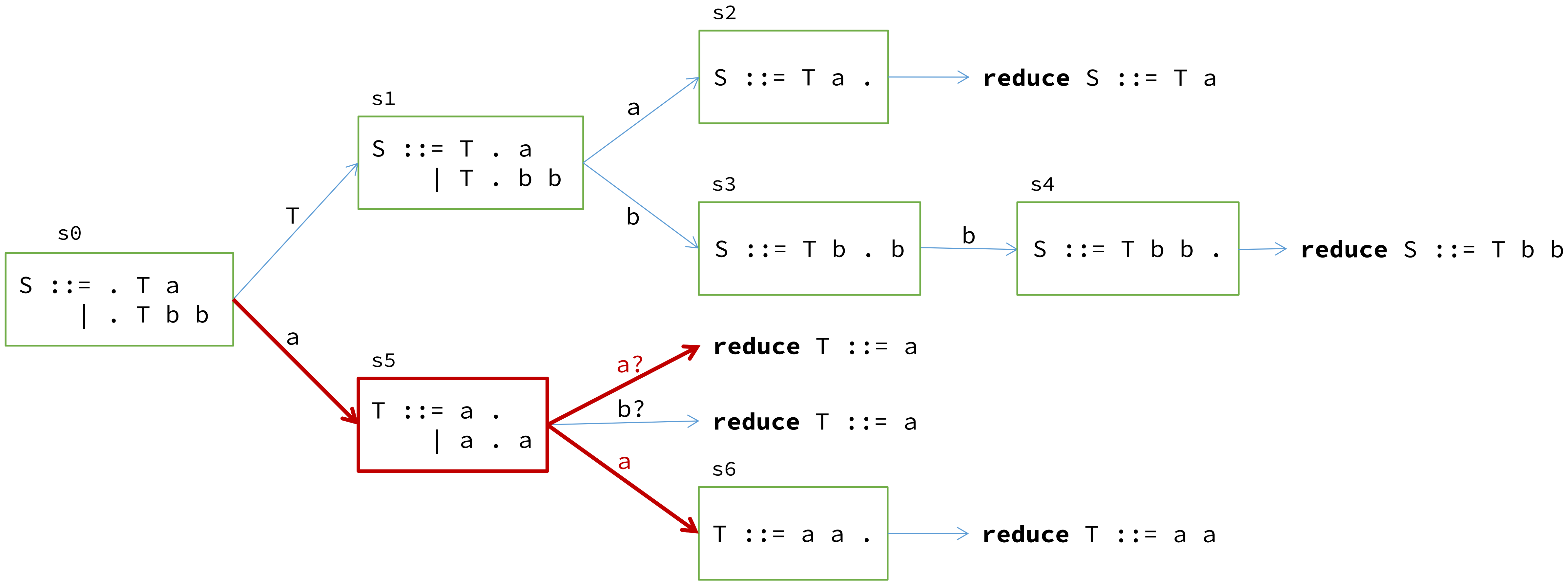
The automaton

It turns into the following LR(1) automaton, with one SHIFT/REDUCE conflict



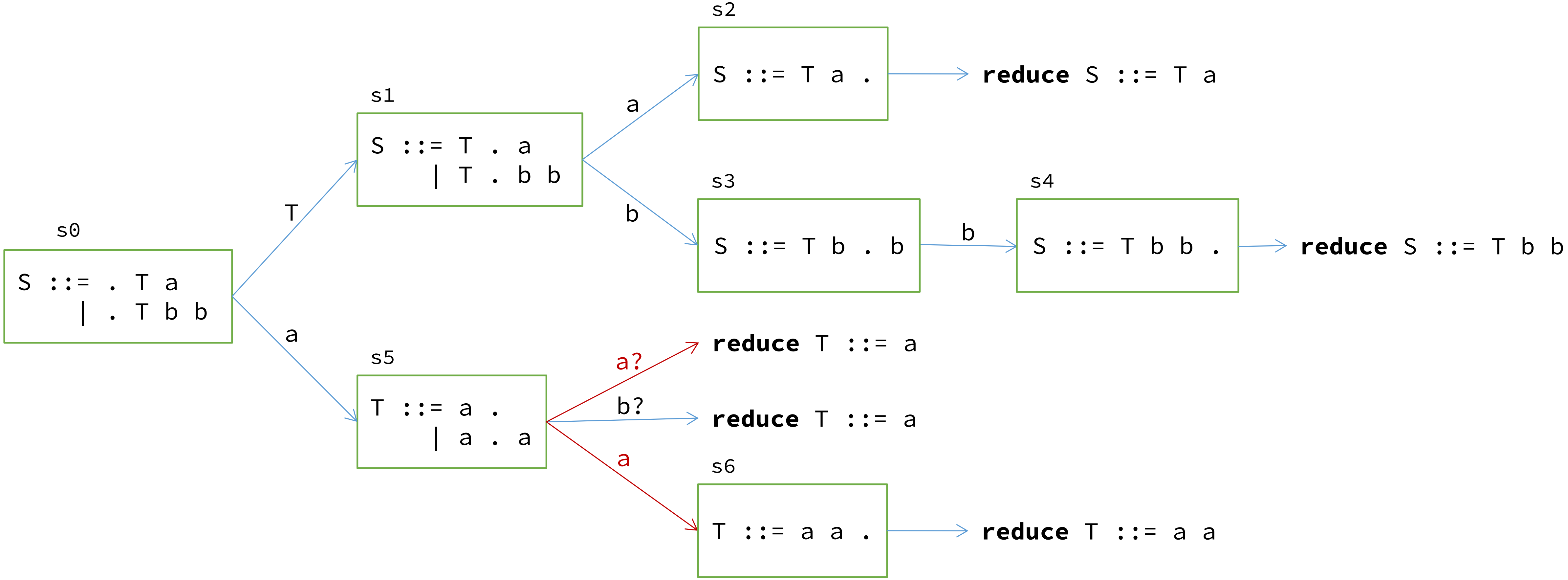
The automaton

It turns into the following LR(1) automaton, with one `SHIFT/REDUCE` conflict



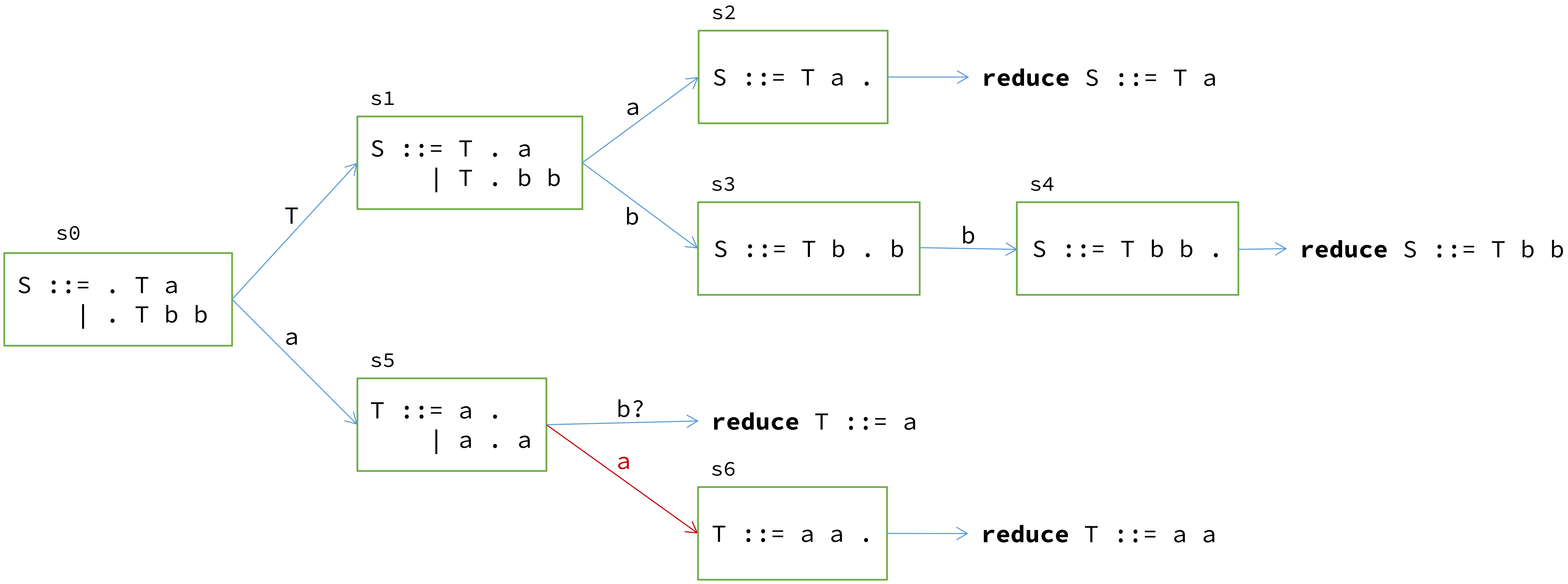
Conflict resolution

Let’s say we decide to `SHIFT`



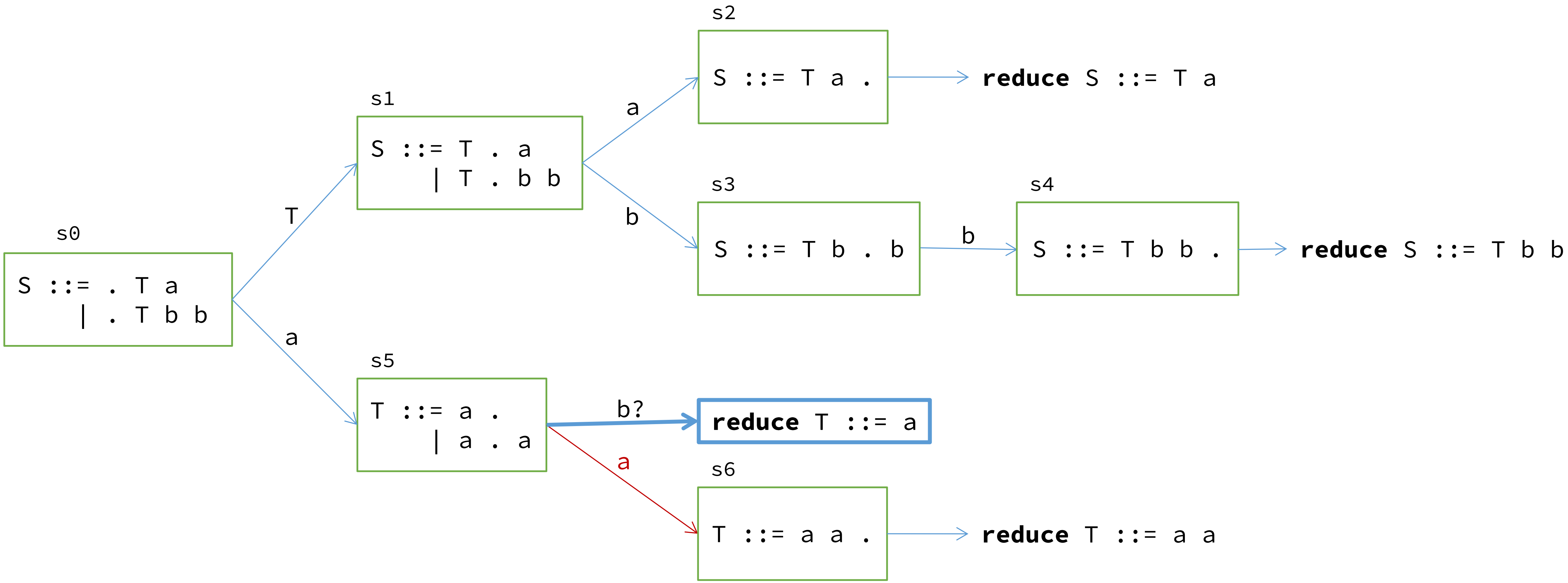
Conflict resolution

Let's say we decide to `SHIFT`



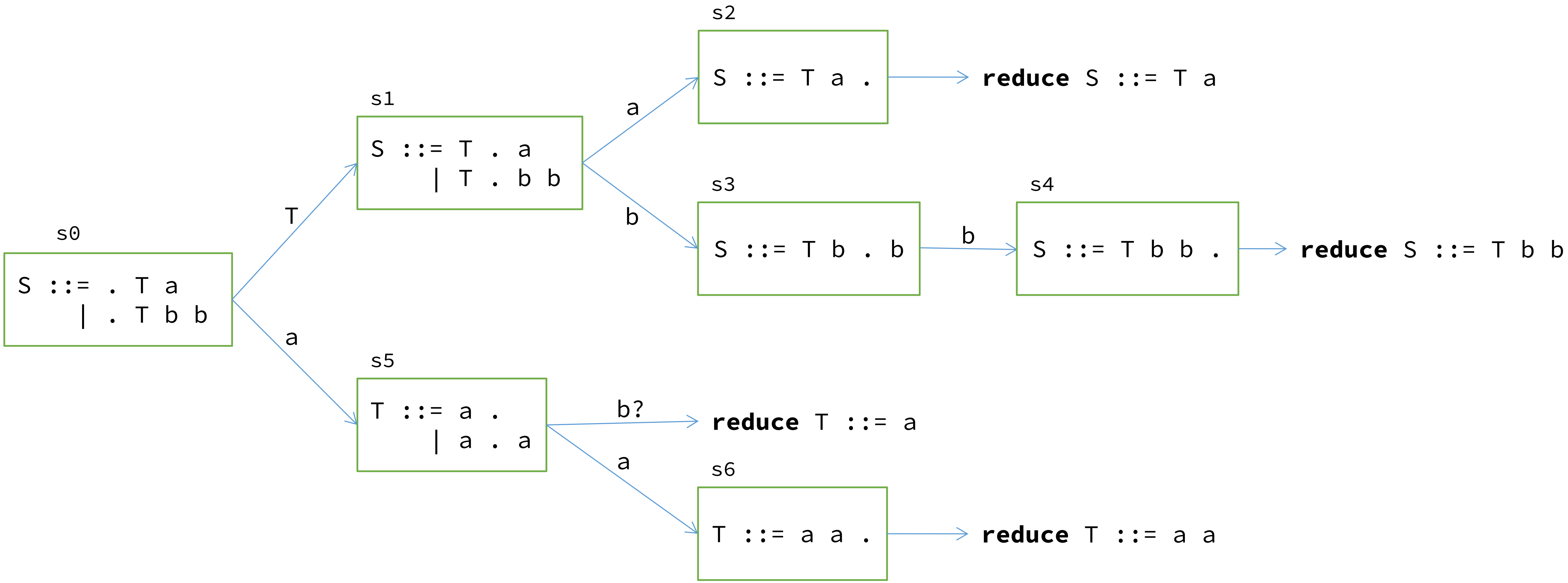
Conflict resolution

Let's say we decide to `SHIFT`



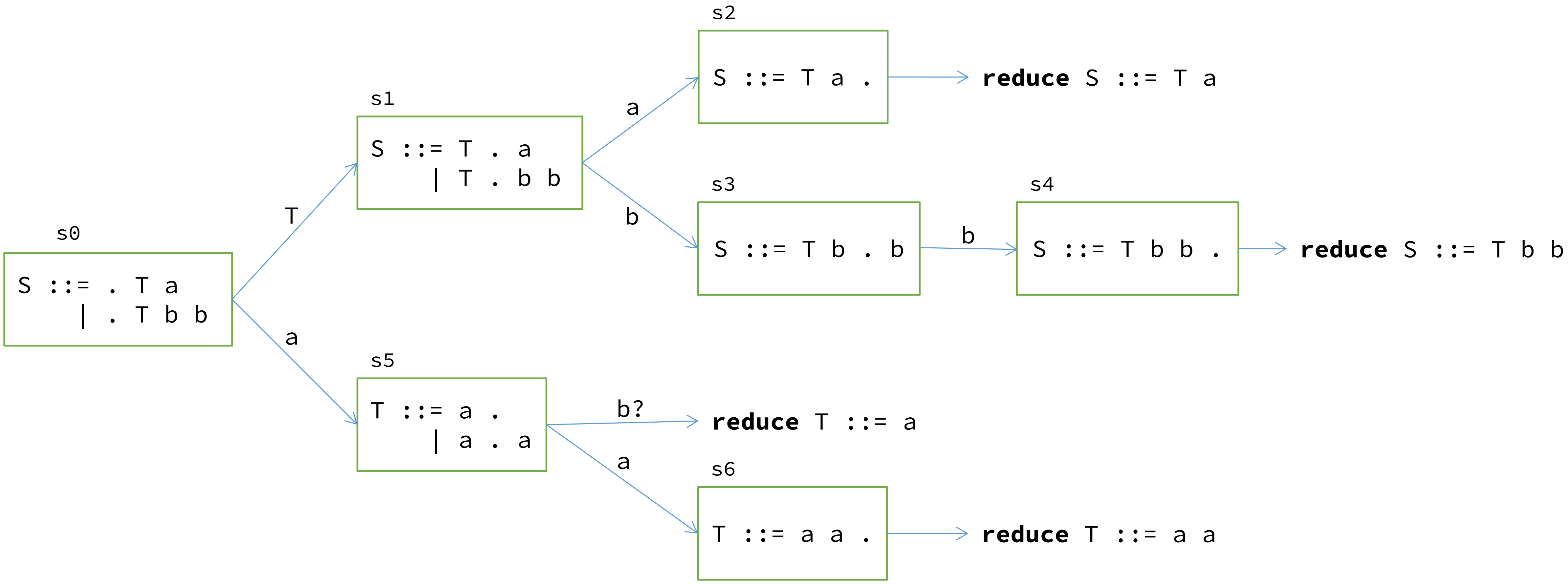
Cost equations

A first attempt at finding costs



Cost equations

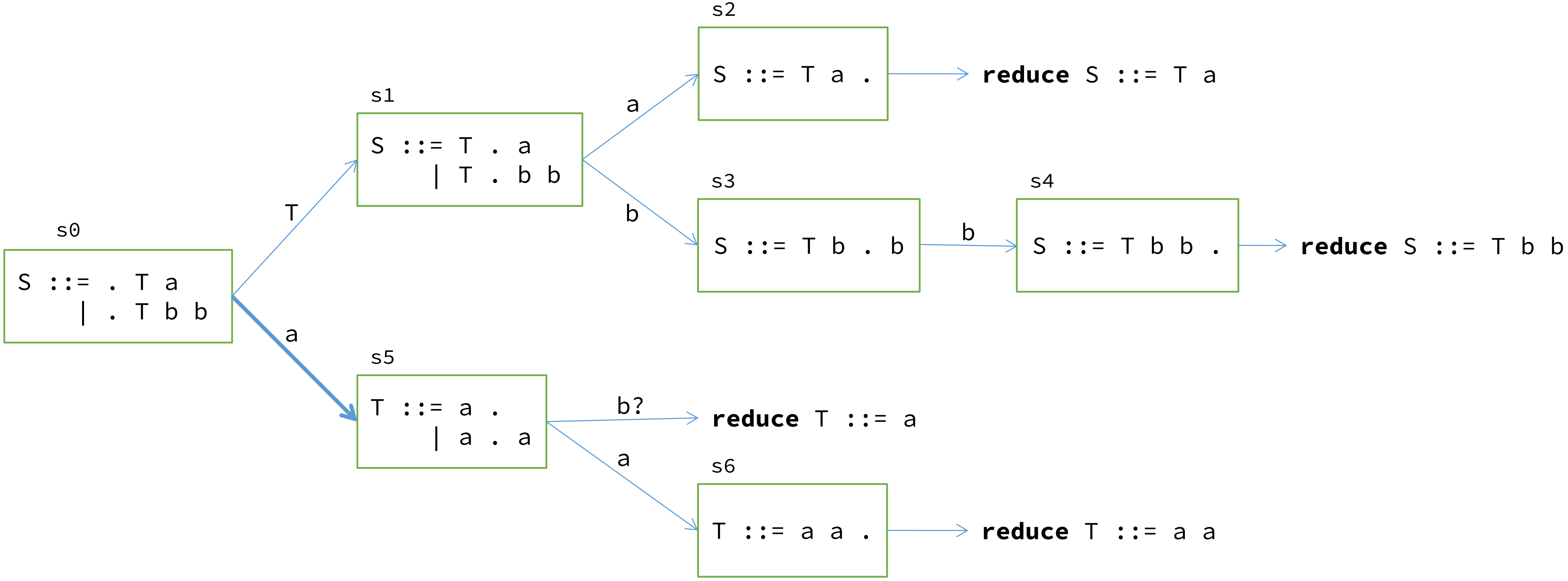
A first attempt at finding costs



Cost equations

$$\mathrm{cost}(\mathrm{s0},\, \mathrm{a}) = 1$$

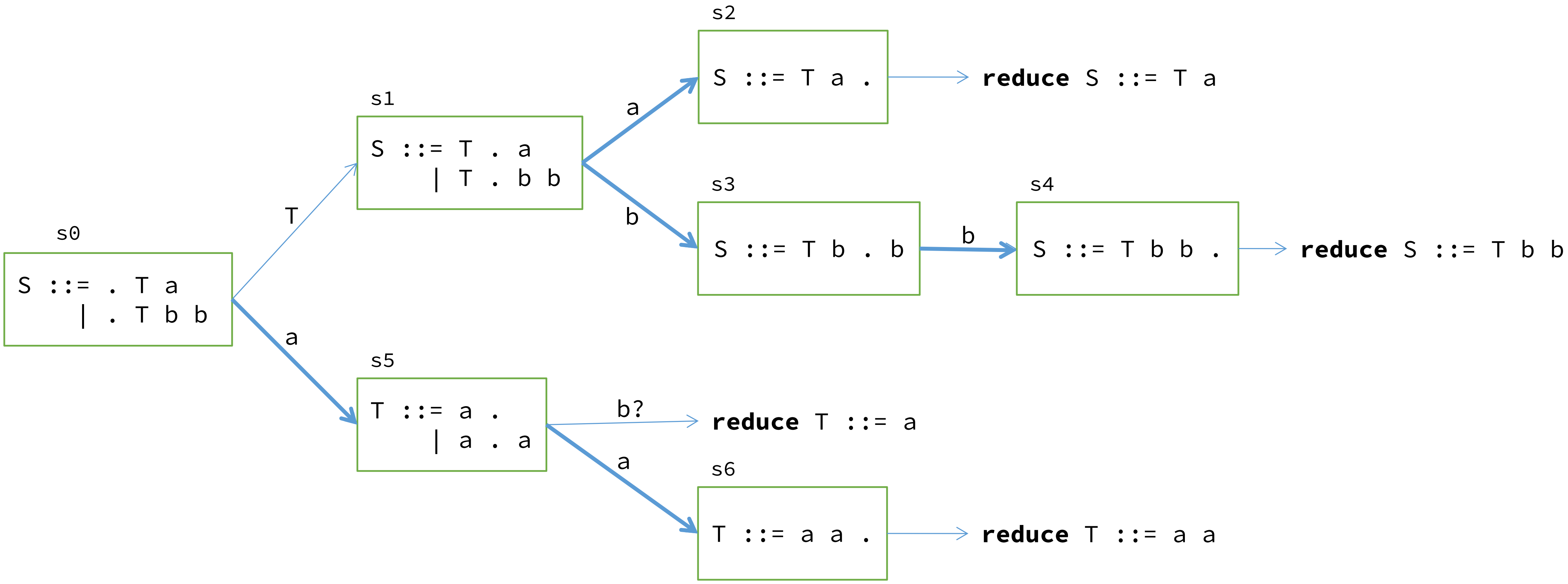
A first attempt at finding costs



Cost equations

$$\begin{aligned}
\text{cost}(\text{s0}, \text{a}) &= 1 \\
\text{cost}(\text{s1}, \text{a}) &= 1 \\
\text{cost}(\text{s5}, \text{a}) &= 1 \\
\text{cost}(\text{s1}, \text{b}) &= 1 \\
\text{cost}(\text{s3}, \text{b}) &= 1
\end{aligned}$$

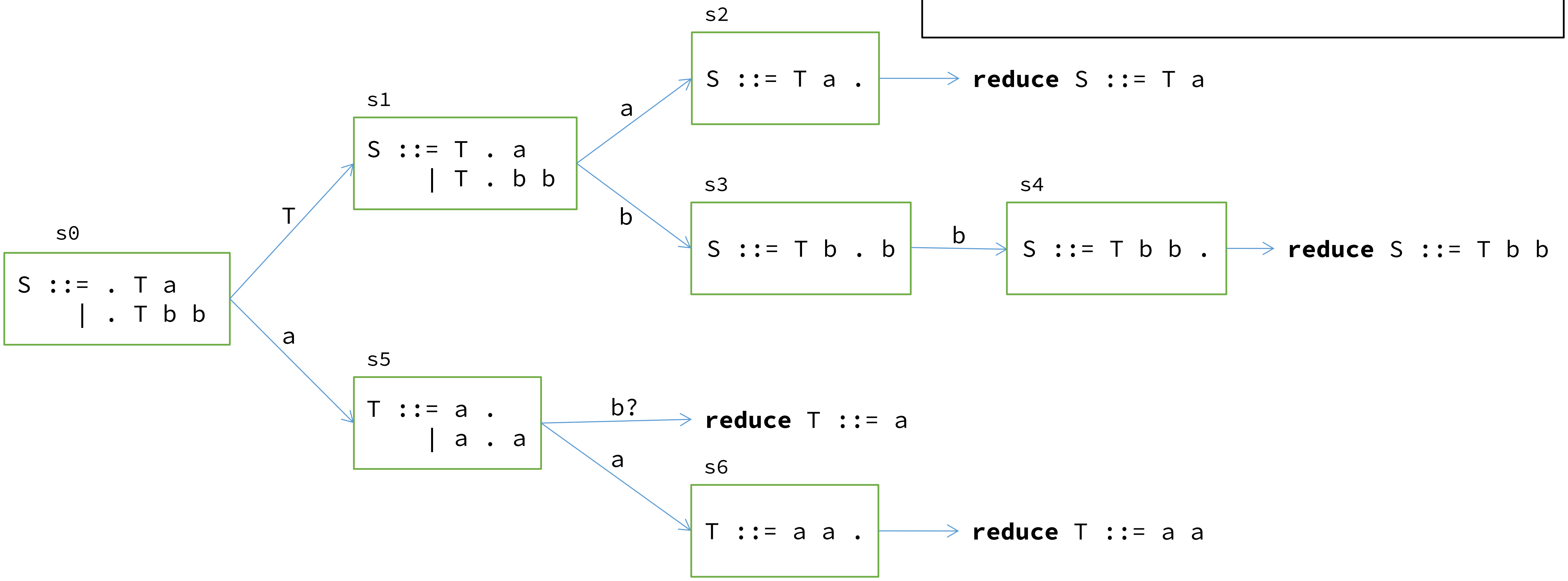
A first attempt at finding costs



Cost equations

$$\begin{aligned}
\text{cost}(s0, a) &= 1 \\
\text{cost}(s1, a) &= 1 \\
\text{cost}(s5, a) &= 1 \\
\text{cost}(s1, b) &= 1 \\
\text{cost}(s3, b) &= 1
\end{aligned}$$

A first attempt at finding costs

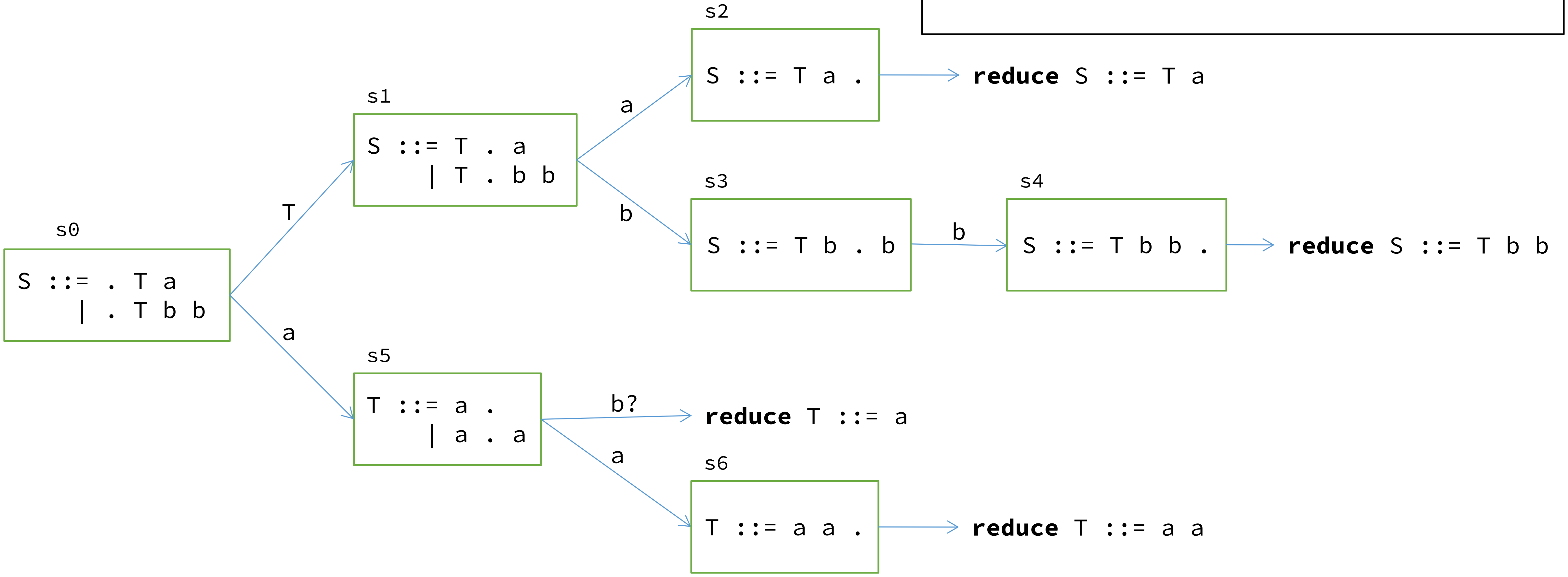


Cost equations

$$\begin{aligned}
\text{cost}(s0, a) &= 1\\
\text{cost}(s1, a) &= 1\\
\text{cost}(s5, a) &= 1\\
\text{cost}(s1, b) &= 1\\
\text{cost}(s3, b) &= 1
\end{aligned}$$

$$\text{cost}(s0, T)$$

A first attempt at finding costs



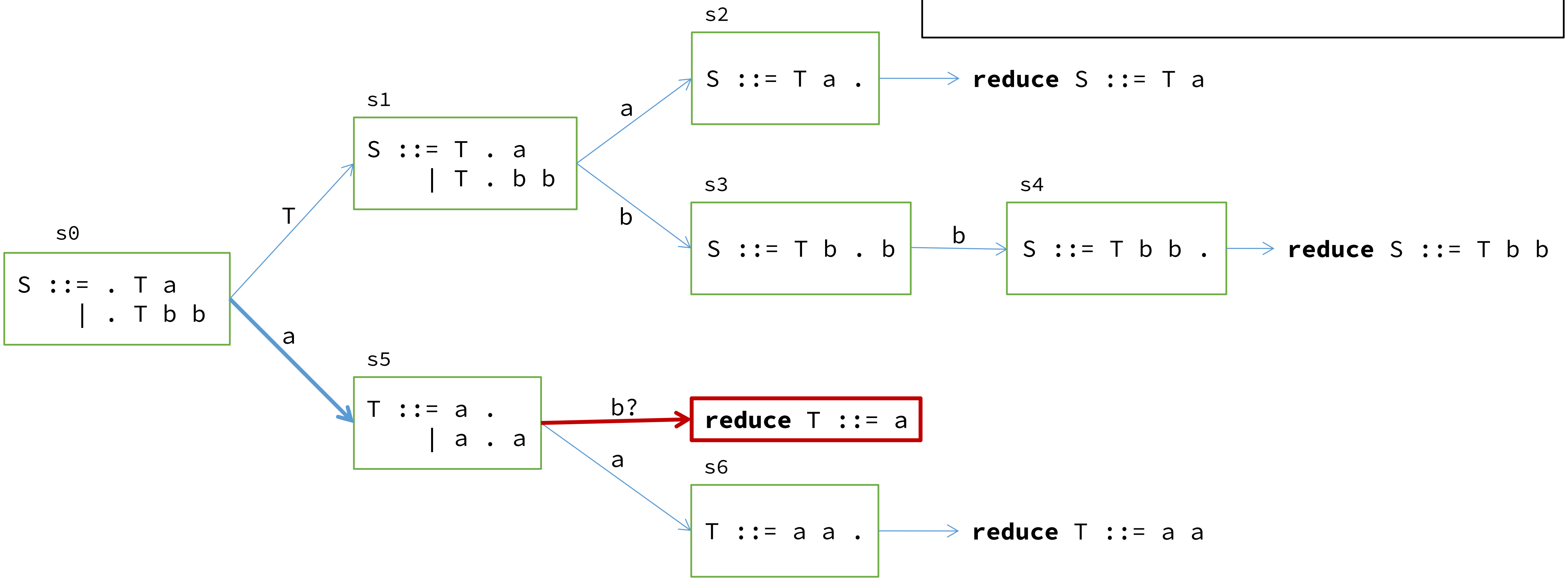
Cost equations

$$\text{cost}(s0, a) = 1$$
$$\text{cost}(s1, a) = 1$$
$$\text{cost}(s5, a) = 1$$
$$\text{cost}(s1, b) = 1$$
$$\text{cost}(s3, b) = 1$$

$\text{cost}(s0, T)$

$\quad\quad \text{cost}(s0, a)$

A first attempt at finding costs



Cost equations

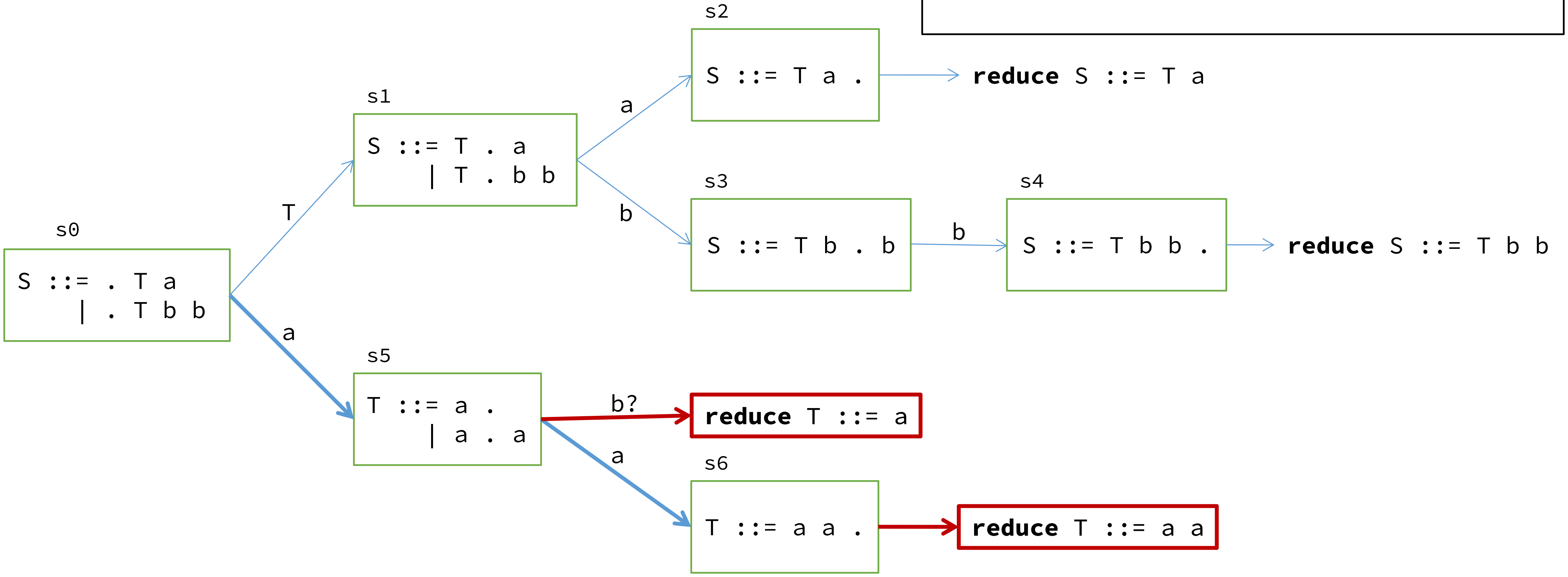
$$\text{cost}(s0, a) = 1$$
$$\text{cost}(s1, a) = 1$$
$$\text{cost}(s5, a) = 1$$
$$\text{cost}(s1, b) = 1$$
$$\text{cost}(s3, b) = 1$$

$\text{cost}(s0, T)$

$\text{cost}(s0, a)$

$\text{cost}(s0, a) + \text{cost}(s5, a)$

A first attempt at finding costs

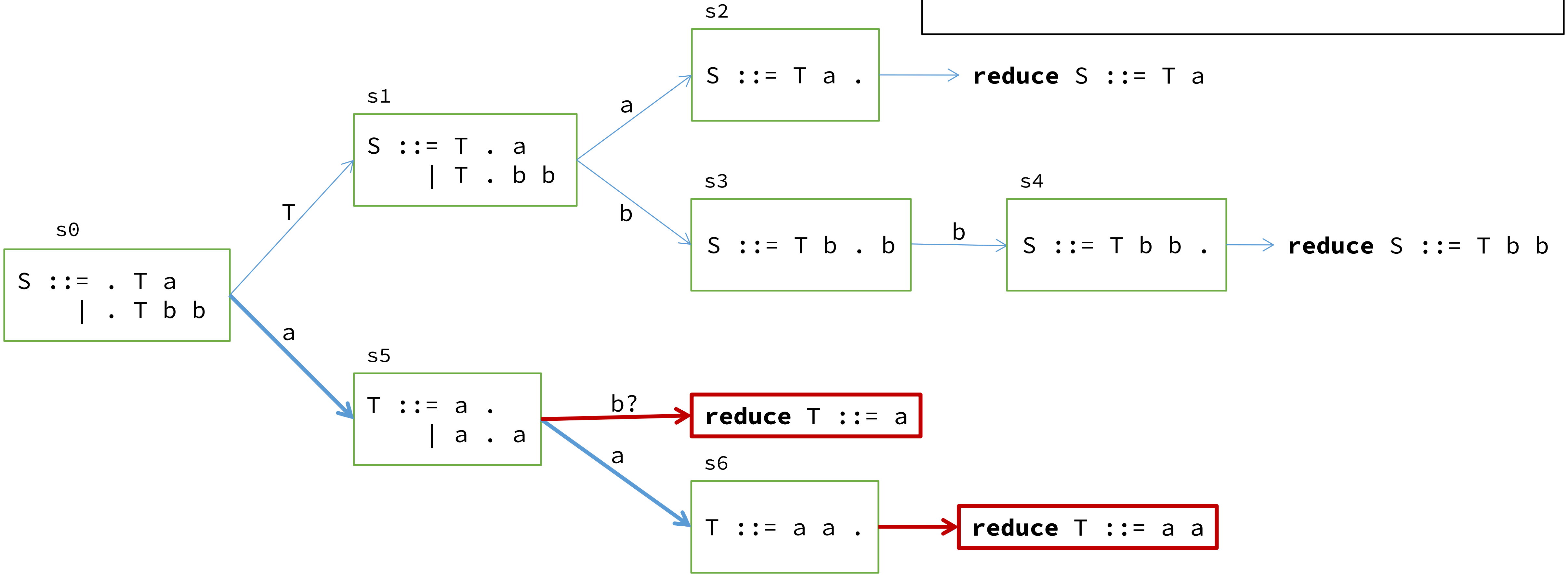


Cost equations

$$\begin{aligned} \text{cost}(s0, a) &= 1 \\ \text{cost}(s1, a) &= 1 \\ \text{cost}(s5, a) &= 1 \\ \text{cost}(s1, b) &= 1 \\ \text{cost}(s3, b) &= 1 \end{aligned}$$

$$\text{cost}(s0, T) = \min \begin{cases} \text{cost}(s0, a) \\ \text{cost}(s0, a) + \text{cost}(s5, a) \end{cases}$$

A first attempt at finding costs

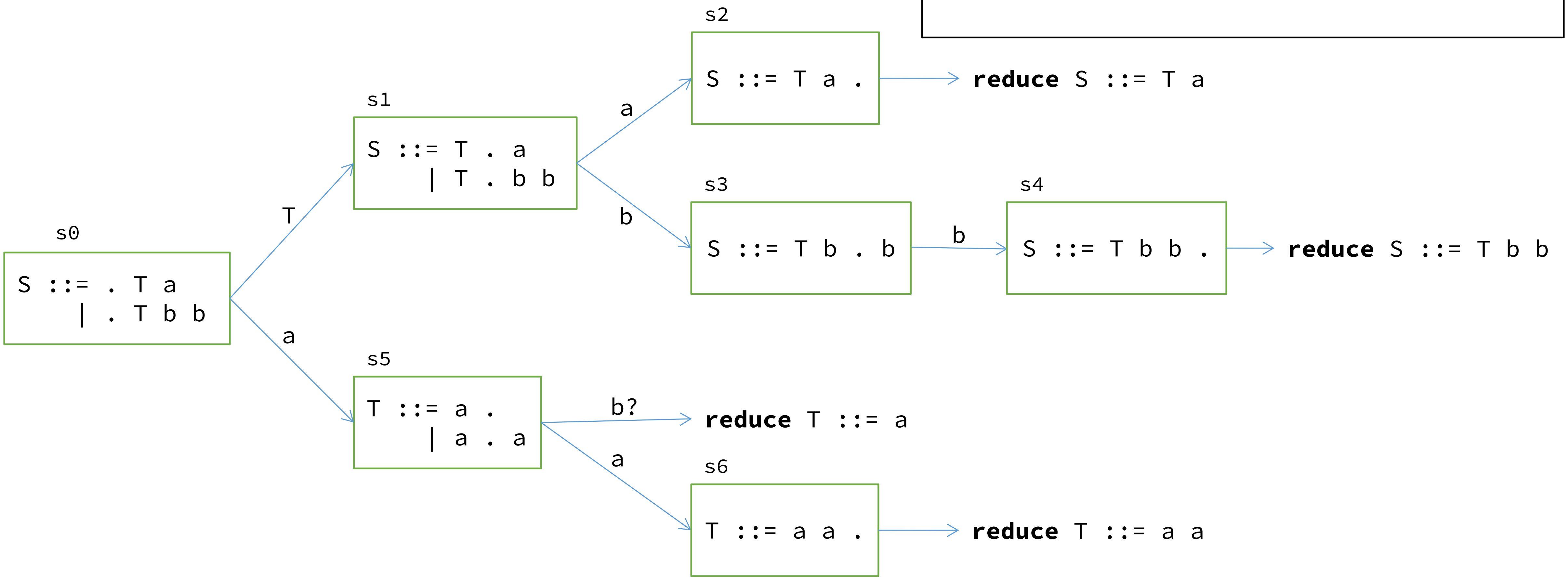


Cost equations

$$\begin{aligned} \text{cost}(s0, a) &= 1 \\ \text{cost}(s1, a) &= 1 \\ \text{cost}(s5, a) &= 1 \\ \text{cost}(s1, b) &= 1 \\ \text{cost}(s3, b) &= 1 \end{aligned}$$

$$\begin{aligned} &\text{cost}(s0, T) \\ &= \min \begin{cases} \text{cost}(s0, a) \\ \text{cost}(s0, a) + \text{cost}(s5, a) \end{cases} \\ &= 1 \end{aligned}$$

A first attempt at finding costs



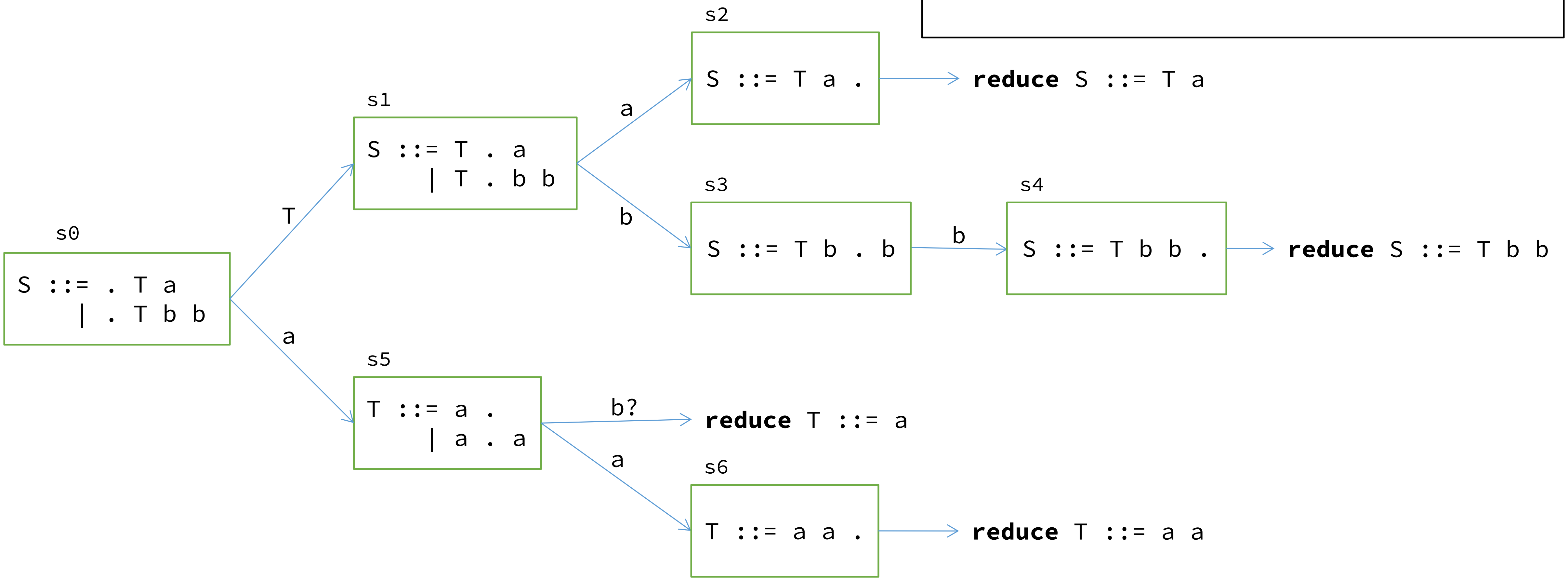
Cost equations

$$\begin{aligned}
\text{cost}(s0, a) &= 1 \\
\text{cost}(s1, a) &= 1 \\
\text{cost}(s5, a) &= 1 \\
\text{cost}(s1, b) &= 1 \\
\text{cost}(s3, b) &= 1
\end{aligned}$$

$$\text{cost}(s0, T) = \min\begin{cases}\text{cost}(s0, a) \\ \text{cost}(s0, a) + \text{cost}(s5, a)\end{cases} = 1$$

$$\text{cost}(s0, S)$$

A first attempt at finding costs



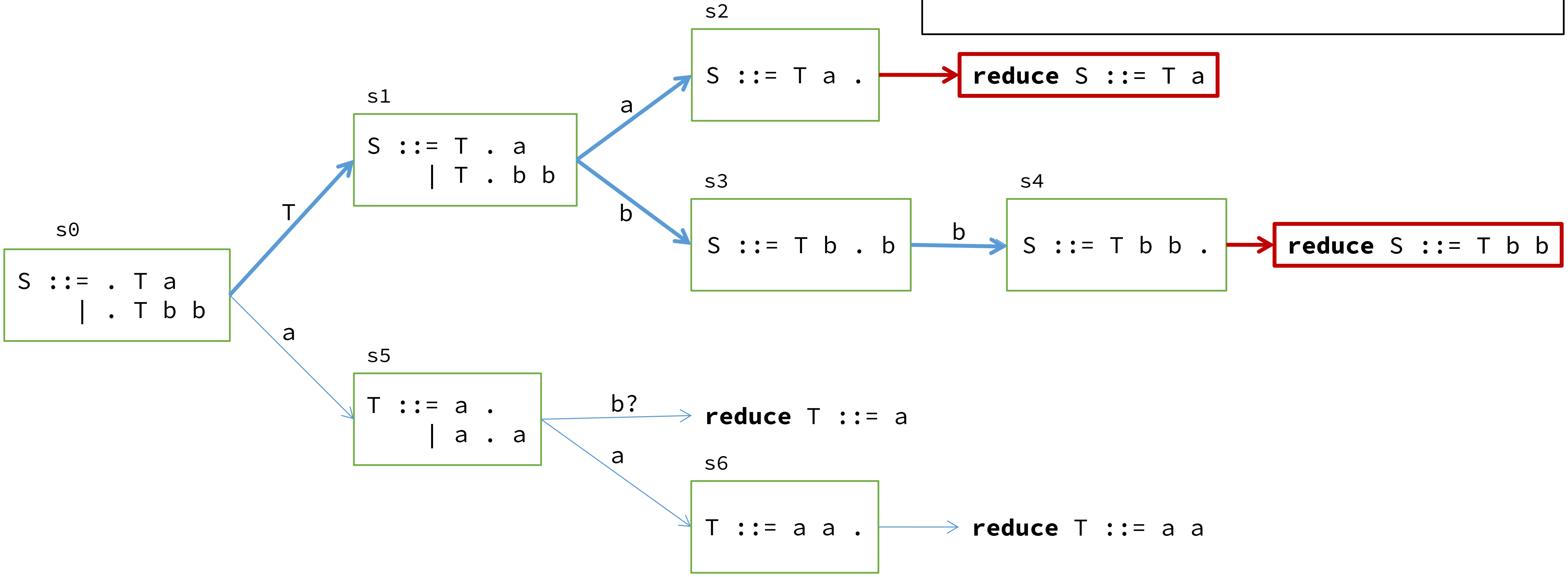
Cost equations

$$\begin{aligned}
\text{cost}(s0, a) &= 1\\
\text{cost}(s1, a) &= 1\\
\text{cost}(s5, a) &= 1\\
\text{cost}(s1, b) &= 1\\
\text{cost}(s3, b) &= 1
\end{aligned}$$

$$\text{cost}(s0, T) = \min\begin{cases}\text{cost}(s0, a)\\ \text{cost}(s0, a) + \text{cost}(s5, a)\end{cases} = 1$$

$$\text{cost}(s0, S) = \min\begin{cases}\text{cost}(s0, T) + \text{cost}(s1, a)\\ \text{cost}(s0, T) + \text{cost}(s1, b) + \text{cost}(s3, b)\end{cases}$$

A first attempt at finding costs



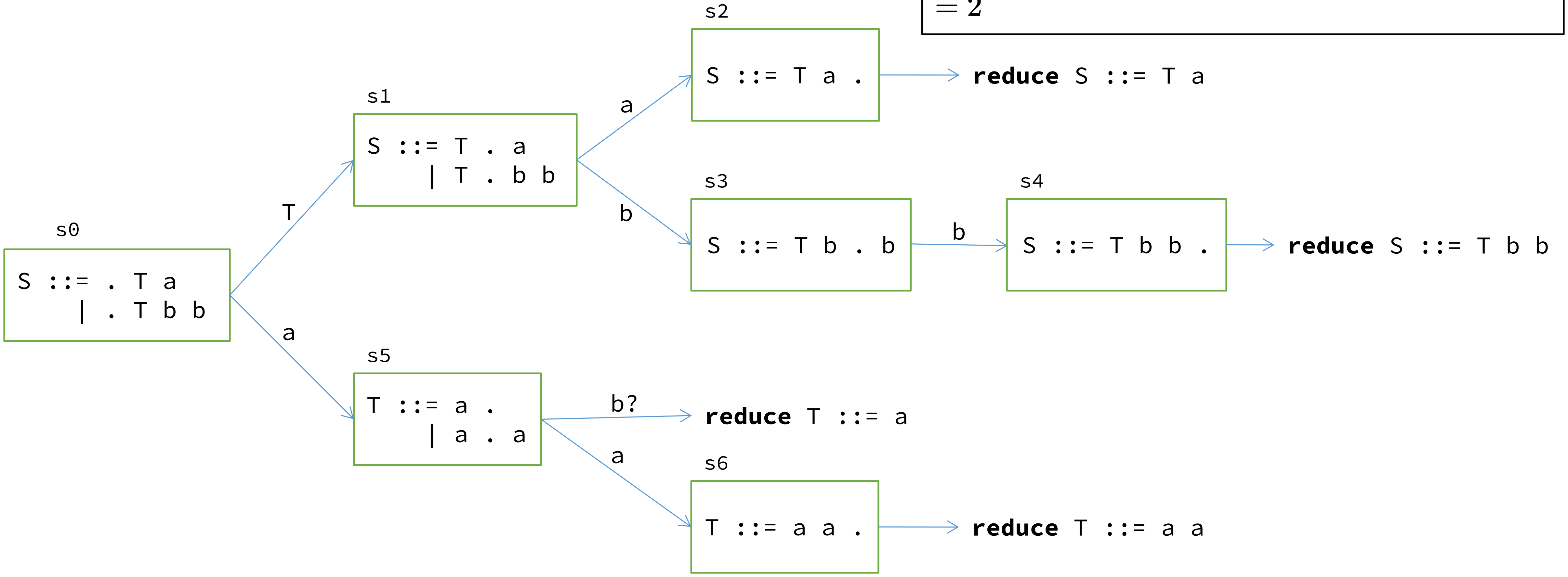
Cost equations

A first attempt at finding costs

$$
\begin{aligned}
\text{cost}(s0, a) &= 1\\
\text{cost}(s1, a) &= 1\\
\text{cost}(s5, a) &= 1\\
\text{cost}(s1, b) &= 1\\
\text{cost}(s3, b) &= 1
\end{aligned}
$$

$$
\begin{aligned}
&\text{cost}(s0, T)\\
&= \min\begin{cases}\text{cost}(s0, a)\\ \text{cost}(s0, a) + \text{cost}(s5, a)\end{cases}\\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
&\text{cost}(s0, S)\\
&= \min\begin{cases}\text{cost}(s0, T) + \text{cost}(s1, a)\\ \text{cost}(s0, T) + \text{cost}(s1, b) + \text{cost}(s3, b)\end{cases}\\
&= 2
\end{aligned}
$$



Cost equations

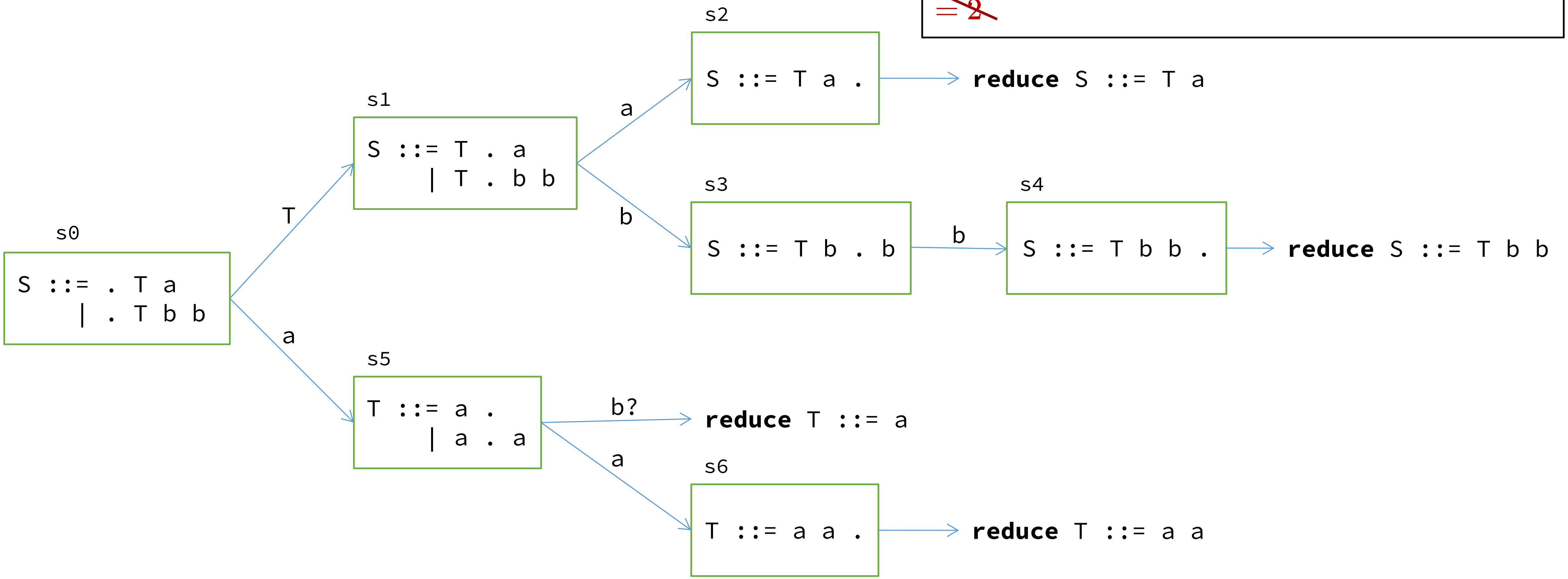
$$
\begin{aligned}
\text{cost}(s0, a) &= 1\\
\text{cost}(s1, a) &= 1\\
\text{cost}(s5, a) &= 1\\
\text{cost}(s1, b) &= 1\\
\text{cost}(s3, b) &= 1
\end{aligned}
$$

$$
\begin{aligned}
&\text{cost}(s0, T)\\
&= \min \begin{cases} \text{cost}(s0, a) \\ \text{cost}(s0, a) + \text{cost}(s5, a) \end{cases}\\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
&\text{cost}(s0, S)\\
&= \min \begin{cases} \text{cost}(s0, T) + \text{cost}(s1, a) \\ \text{cost}(s0, T) + \text{cost}(s1, b) + \text{cost}(s3, b) \end{cases}
\end{aligned}
$$

~~$= 2$~~

A first attempt at finding costs



Cost equations

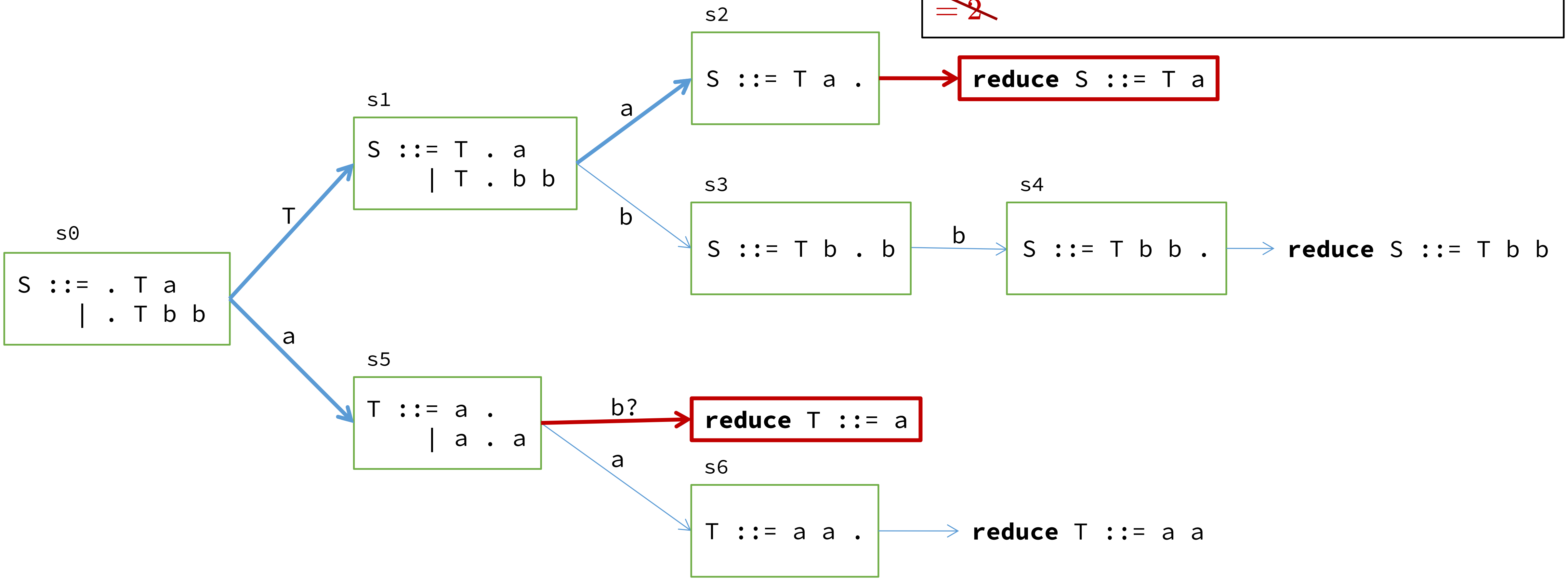
$$
\begin{aligned}
\text{cost}(s0, a) &= 1\\
\text{cost}(s1, a) &= 1\\
\text{cost}(s5, a) &= 1\\
\text{cost}(s1, b) &= 1\\
\text{cost}(s3, b) &= 1
\end{aligned}
$$

$$
\begin{aligned}
&\text{cost}(s0, T)\\
&= \min \begin{cases} \text{cost}(s0, a) \\ \text{cost}(s0, a) + \text{cost}(s5, a) \end{cases}\\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
&\text{cost}(s0, S)\\
&= \min \begin{cases} \text{cost}(s0, T) + \text{cost}(s1, a) \\ \text{cost}(s0, T) + \text{cost}(s1, b) + \text{cost}(s3, b) \end{cases}
\end{aligned}
$$

~~$= 2$~~

A first attempt at finding costs



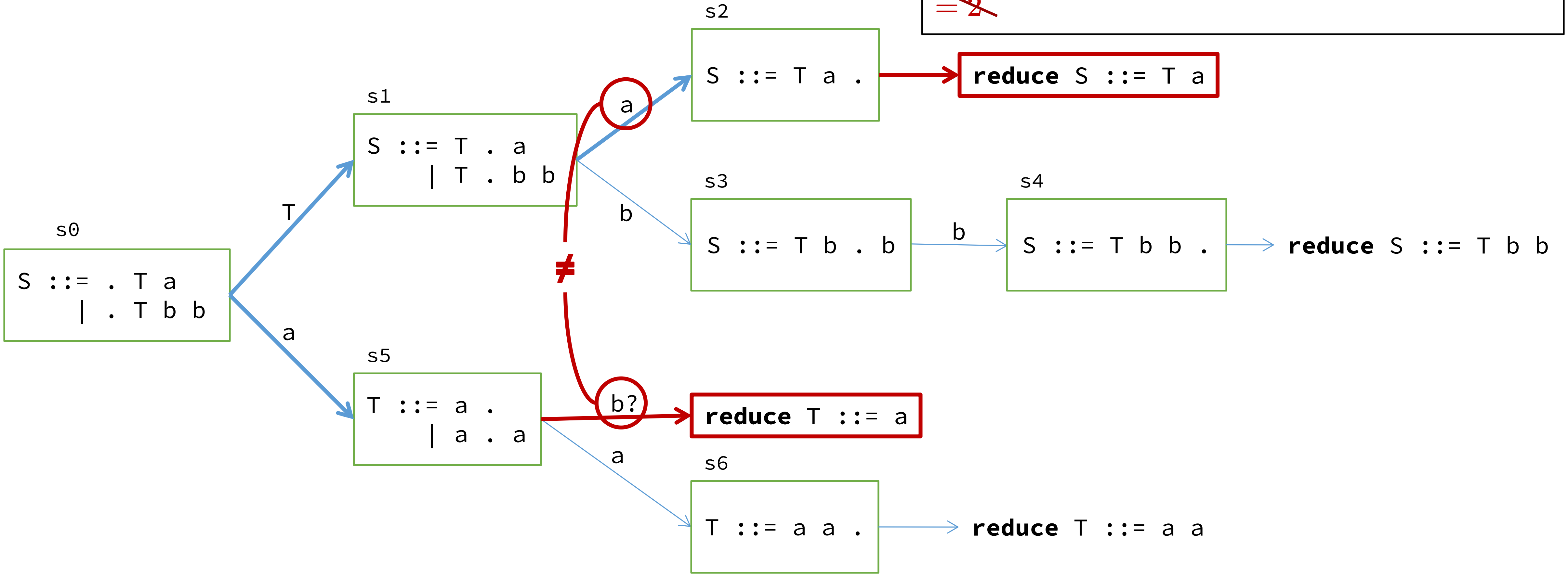
Cost equations

A first attempt at finding costs

$$\text{cost}(s0, a) = 1$$
$$\text{cost}(s1, a) = 1$$
$$\text{cost}(s5, a) = 1$$
$$\text{cost}(s1, b) = 1$$
$$\text{cost}(s3, b) = 1$$

$$\text{cost}(s0, T) = \min \begin{cases} \text{cost}(s0, a) \\ \text{cost}(s0, a) + \text{cost}(s5, a) \end{cases} = 1$$

$$\text{cost}(s0, S) = \min \begin{cases} \text{cost}(s0, T) + \text{cost}(s1, a) \\ \text{cost}(s0, T) + \text{cost}(s1, b) + \text{cost}(s3, b) \end{cases} \not= 2$$



Cost equations

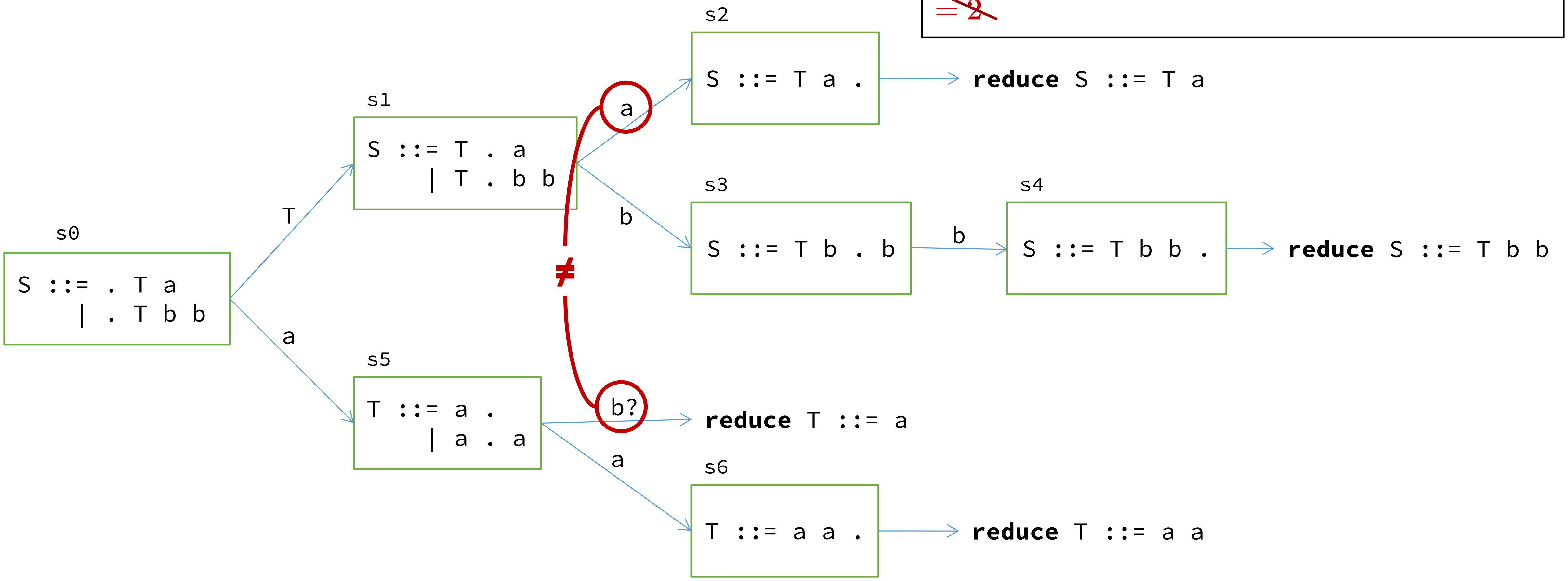
$$\mathrm{cost}(s0, a) = 1$$
$$\mathrm{cost}(s1, a) = 1$$
$$\mathrm{cost}(s5, a) = 1$$
$$\mathrm{cost}(s1, b) = 1$$
$$\mathrm{cost}(s3, b) = 1$$

$$\mathrm{cost}(s0, T) = \min \begin{cases} \mathrm{cost}(s0, a) \\ \mathrm{cost}(s0, a) + \mathrm{cost}(s5, a) \end{cases} = 1$$

$$\mathrm{cost}(s0, S) = \min \begin{cases} \mathrm{cost}(s0, T) + \mathrm{cost}(s1, a) \\ \mathrm{cost}(s0, T) + \mathrm{cost}(s1, b) + \mathrm{cost}(s3, b) \end{cases}$$

~~= 2~~

A first attempt at finding costs



Cost matrices in the $(\min, +)$ semiring

A single integer per edge is not sufficient to carry the cost information.

We use **matrices indexed by terminals**:

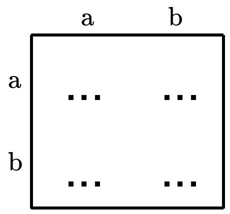
- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition

Cost matrices in the $(\min, +)$ semiring

A single integer per edge is not sufficient to carry the cost information.

We use **matrices indexed by terminals**:

- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition

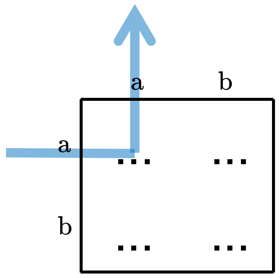


Cost matrices in the $(\min, +)$ semiring

A single integer per edge is not sufficient to carry the cost information.

We use **matrices indexed by terminals**:

- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition

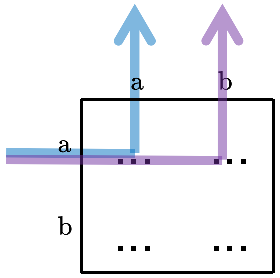


Cost matrices in the $(\min, +)$ semiring

A single integer per edge is not sufficient to carry the cost information.

We use **matrices indexed by terminals**:

- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition

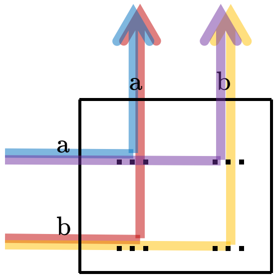


Cost matrices in the $(\min, +)$ semiring

A single integer per edge is not sufficient to carry the cost information.

We use **matrices indexed by terminals**:

- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition



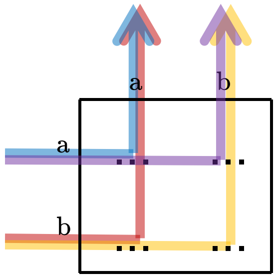
Cost matrices in the $(\min, +)$ semiring

A single integer per edge is not sufficient to carry the cost information.

We use **matrices indexed by terminals**:

- the **row** index represents the **lookahead token before** taking the transition
- the **column** index represents the **lookahead token after** taking the transition

In the $(\min, +)$ semiring, **matrix product** represents the cost of a **sequence**.



LR(1) matrix-based cost equations

Computing costs with matrices:

LR(1) matrix-based cost equations

Computing costs with matrices:

$$\mathrm{cost}(\mathrm{s0},\ \mathrm{a}) = \mathrm{cost}(\mathrm{s1},\ \mathrm{a}) = \mathrm{cost}(\mathrm{s5},\ \mathrm{a}) =$$

LR(1) matrix-based cost equations

Computing costs with matrices:

$$\text{cost}(\text{s0}, \text{a}) = \text{cost}(\text{s1}, \text{a}) = \text{cost}(\text{s5}, \text{a}) = \begin{array}{c|cc|} & \text{a} & \text{b} \\ \hline \text{a} & 1 & 1 \\ \text{b} & \infty & \infty \\ \hline \end{array}$$

LR(1) matrix-based cost equations

Computing costs with matrices:

$$\text{cost}(\text{s0, a}) = \text{cost}(\text{s1, a}) = \text{cost}(\text{s5, a}) = \begin{array}{c|cc} & \text{a} & \text{b} \\ \hline \text{a} & 1 & 1 \\ \text{b} & \infty & \infty \end{array}$$

$$\text{cost}(\text{s1, b}) = \text{cost}(\text{s3, b}) = \begin{array}{c|cc} & \text{a} & \text{b} \\ \hline \text{a} & \infty & \infty \\ \text{b} & 1 & 1 \end{array}$$

LR(1) matrix-based cost equations

Computing costs with matrices:

$$\text{cost}(s0, a) = \text{cost}(s1, a) = \text{cost}(s5, a) = \begin{array}{c|cc} & a & b \\ \hline a & 1 & 1 \\ b & \infty & \infty \end{array}$$

$$\text{cost}(s1, b) = \text{cost}(s3, b) = \begin{array}{c|cc} & a & b \\ \hline a & \infty & \infty \\ b & 1 & 1 \end{array}$$

$$\text{cost}(s0, T) = \min \begin{cases} \text{cost}(s0, a) \\ \text{cost}(s0, a) \cdot \text{cost}(s5, a) \end{cases} = \begin{array}{c|cc} & a & b \\ \hline a & 2 & 1 \\ b & \infty & \infty \end{array}$$

LR(1) matrix-based cost equations

Computing costs with matrices:

$$\text{cost}(s0, a) = \text{cost}(s1, a) = \text{cost}(s5, a) = \begin{array}{c|cc|} & a & b \\ \hline a & 1 & 1 \\ b & \infty & \infty \\ \hline \end{array}$$

$$\text{cost}(s1, b) = \text{cost}(s3, b) = \begin{array}{c|cc|} & a & b \\ \hline a & \infty & \infty \\ b & 1 & 1 \\ \hline \end{array}$$

$$\text{cost}(s0, T) = \min \begin{cases} \text{cost}(s0, a) \\ \text{cost}(s0, a) \cdot \text{cost}(s5, a) \end{cases} = \begin{array}{c|cc|} & a & b \\ \hline a & 2 & 1 \\ b & \infty & \infty \\ \hline \end{array}$$

$$\text{cost}(s0, S) = \min \begin{cases} \text{cost}(s0, T) \cdot \text{cost}(s1, a) \\ \text{cost}(s0, T) \cdot \text{cost}(s1, b) \cdot \text{cost}(s3, b) \end{cases} = \begin{array}{c|cc|} & a & b \\ \hline a & 3 & 3 \\ b & \infty & \infty \\ \hline \end{array}$$

Big matrices?

A $|\mathrm{T}| \times |\mathrm{T}|$ matrix for each transition and reduction step consumes a lot of space.

Big matrices?

A $|\mathrm{T}| \times |\mathrm{T}|$ matrix for each transition and reduction step consumes a lot of space.

It is often wasteful: lookahead terminals often behave the same.
This leads to **identical columns** in the cost matrix.

Big matrices?

A $|\mathtt{T}| \times |\mathtt{T}|$ matrix for each transition and reduction step consumes a lot of space.

It is often wasteful: lookahead terminals often behave the same.
This leads to **identical columns** in the cost matrix.

We can characterize and group lookahead tokens with identical behavior.

Classifying terminals

- The goto transition $(\mathrm{s0}, \mathrm{T})$ is followed when one of its productions is reduced:

Classifying terminals

- The goto transition $(\mathrm{s0}, \mathrm{T})$ is followed when one of its productions is reduced:
  - `T := a` reduced when lookahead is `b` $\rightarrow$ $\{$`b`$\}$

Classifying terminals

- The goto transition $(\mathrm{s0}, \mathrm{T})$ is followed when one of its productions is reduced:
  - `T := a` reduced when lookahead is `b` → {b}
  - `T := a a` always reduced → {a,b}

Classifying terminals

- The goto transition $(s0, T)$ is followed when one of its productions is reduced:
  - `T := a` reduced when lookahead is b → {b}
  - `T := a a` always reduced → {a,b}

- The **cases to consider** are given by the **coarsest refinement** of {a,b} and {b}:

  {{a},{b}}

Classifying terminals

- The goto transition $(\mathrm{s0}, \mathrm{T})$ is followed when one of its productions is reduced:
  - `T := a` reduced when lookahead is b $\rightarrow$ {b}
  - `T := a a` always reduced $\rightarrow$ {a,b}

- The **cases to consider** are given by the **coarsest refinement** of {a,b} and {b}:

  {{a},{b}}

  {T := a a}

Classifying terminals

- The goto transition $(\mathrm{s0}, \mathrm{T})$ is followed when one of its productions is reduced:
  - `T := a` reduced when lookahead is b $\rightarrow$ $\{b\}$
  - `T := a a` always reduced $\rightarrow$ $\{a,b\}$

- The **cases to consider** are given by the **coarsest refinement** of $\{a,b\}$ and $\{b\}$:

$\{\{a\},\{b\}\}$

{`T := a a`}   {`T := a`, `T := a a`}

Classifying terminals

- The goto transition $(\mathrm{s0}, \mathrm{T})$ is followed when one of its productions is reduced:
  - `T := a` reduced when lookahead is b $\rightarrow$ {b}
  - `T := a a` always reduced $\rightarrow$ {a,b}

- The **cases to consider** are given by the **coarsest refinement** of {a,b} and {b}:

{{a},{b}}

{T := a a}   {T := a, T := a a}

Before starting to compute costs, we know what lookahead symbols to distinguish!

Idea #2: compacting matrices

Compacting columns

Let's assume that:

- we want to compact a matrix m
- we have 4 terminals, a, b, c and d
- our characterization found the partition $\{\{\mathrm{a,b}\}, \{\mathrm{c}\}, \{\mathrm{d}\}\}$.

$\mathrm{m} =$

	a	b	c	d
a	1	1	2	3
b	∞	∞	∞	∞
c	∞	∞	3	∞
d	1	1	1	1

Compacting columns

m =

	a	b	c	d
a	1	1	2	3
b	∞	∞	∞	∞
c	∞	∞	3	∞
d	1	1	1	1

Compacting columns

m =

	a	b	c	d
a	1	1	2	3
b	∞	∞	∞	∞
c	∞	∞	3	∞
d	1	1	1	1

Compacting columns

m =

	a	b	c	d
a	1	1	2	3
b	∞	∞	∞	∞
c	∞	∞	3	∞
d	1	1	1	1

Compacting columns

m =

	{a, b}	{c}	{d}
a	1	2	3
b	∞	∞	∞
c	∞	3	∞
d	1	1	1

Compacting columns

m =

	$\{a, b\}$	$\{c\}$	$\{d\}$
a	1	2	3
b	∞	∞	∞
c	∞	3	∞
d	1	1	1

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

$\cdot$

	a	b	c	d
a	n_{aa}	...	...	...
b	n_{ba}	...	...	...
c	n_{ca}	...	...	...
d	n_{da}	...	...	...

$= r$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	…	…
b	…	…	…
c	…	…	…
d	…	…	…

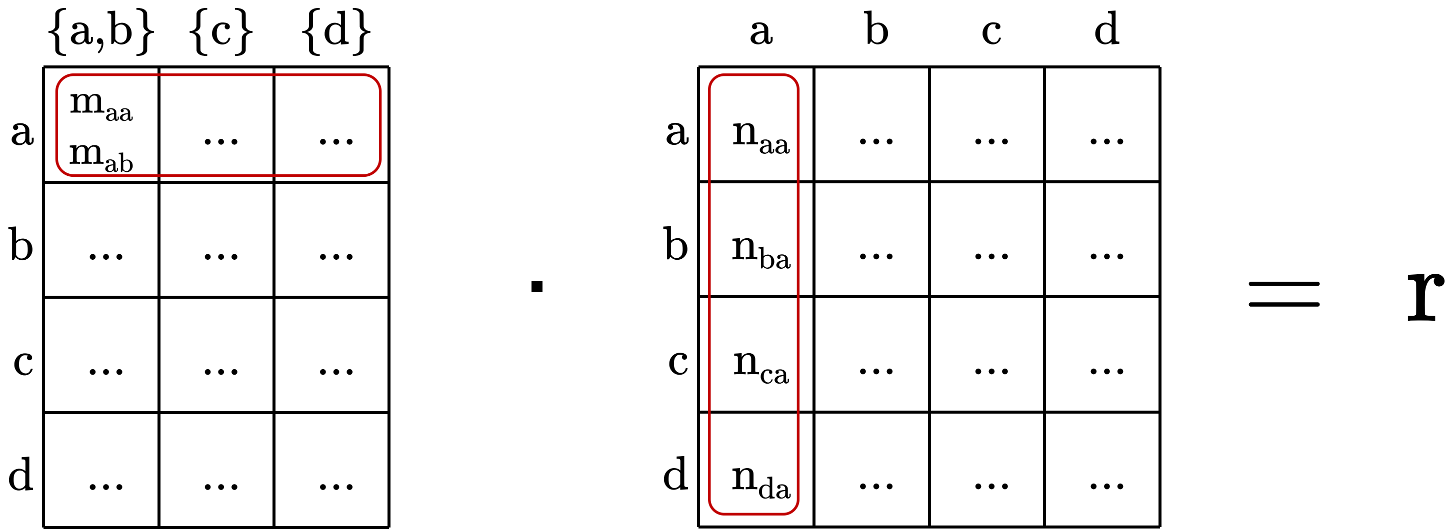
$\cdot$

	a	b	c	d
a	n_{aa}	…	…	…
b	n_{ba}	…	…	…
c	n_{ca}	…	…	…
d	n_{da}	…	…	…

$= r$

$r_{aa} =$

Compacting rows



$$r_{aa} = (m_{aa} + n_{aa}) \wedge (m_{ab} + n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

$$x \wedge y = \min\{x,y\}$$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

$\cdot$

	a	b	c	d
a	n_{aa}	...	...	...
b	n_{ba}	...	...	...
c	n_{ca}	...	...	...
d	n_{da}	...	...	...

$= r$

$$r_{aa} = (m_{aa} + n_{aa}) \wedge (m_{ab} + n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

$\cdot$

	a	b	c	d
a	n_{aa}	...	...	...
b	n_{ba}	...	...	...
c	n_{ca}	...	...	...
d	n_{da}	...	...	...

$= \ r$

$$r_{aa} = (m_{aa} + n_{aa}) \wedge (m_{ab} + n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

$$(m_{aa} + n_{aa}) \wedge (m_{ab} + n_{ba}) = m_{aa} + (n_{aa} \wedge n_{ba})$$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

$\cdot$

	a	b	c	d
a	n_{aa}	...	...	...
b	n_{ba}	...	...	...
c	n_{ca}	...	...	...
d	n_{da}	...	...	...

$= r$

$$r_{aa} = (m_{aa} + n_{aa}) \wedge (m_{ab} + n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

$$r_{aa} = (m_{aa} + n_{aa} \wedge n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

$\cdot$

	a	b	c	d
a	n_{aa}	...	...	...
b	n_{ba}	...	...	...
c	n_{ca}	...	...	...
d	n_{da}	...	...	...

$= \; r$

$$r_{aa} = (m_{aa} + n_{aa}) \wedge (m_{ab} + n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

$$r_{aa} = (m_{aa} + \boxed{n_{aa} \wedge n_{ba}}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

$\cdot$

	a	b	c	d
a	n_{aa}	...	...	...
b	n_{ba}	...	...	...
c	n_{ca}	...	...	...
d	n_{da}	...	...	...

$= \; r$

$$r_{aa} = (m_{aa} + n_{aa} \wedge n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

$\cdot$

	a	b	c	d
{a,b}	$n_{aa} \wedge n_{ba}$	... $\wedge$ ...	... $\wedge$ ...	... $\wedge$ ...
{c}	n_{ca}	...	...	...
{d}	n_{da}	...	...	...

$= \mathrm{r}$

$$r_{aa} = (m_{aa} + n_{aa} \wedge n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

Compacting rows

	{a,b}	{c}	{d}
a	m_{aa} m_{ab}	...	...
b	...	...	...
c	...	...	...
d	...	...	...

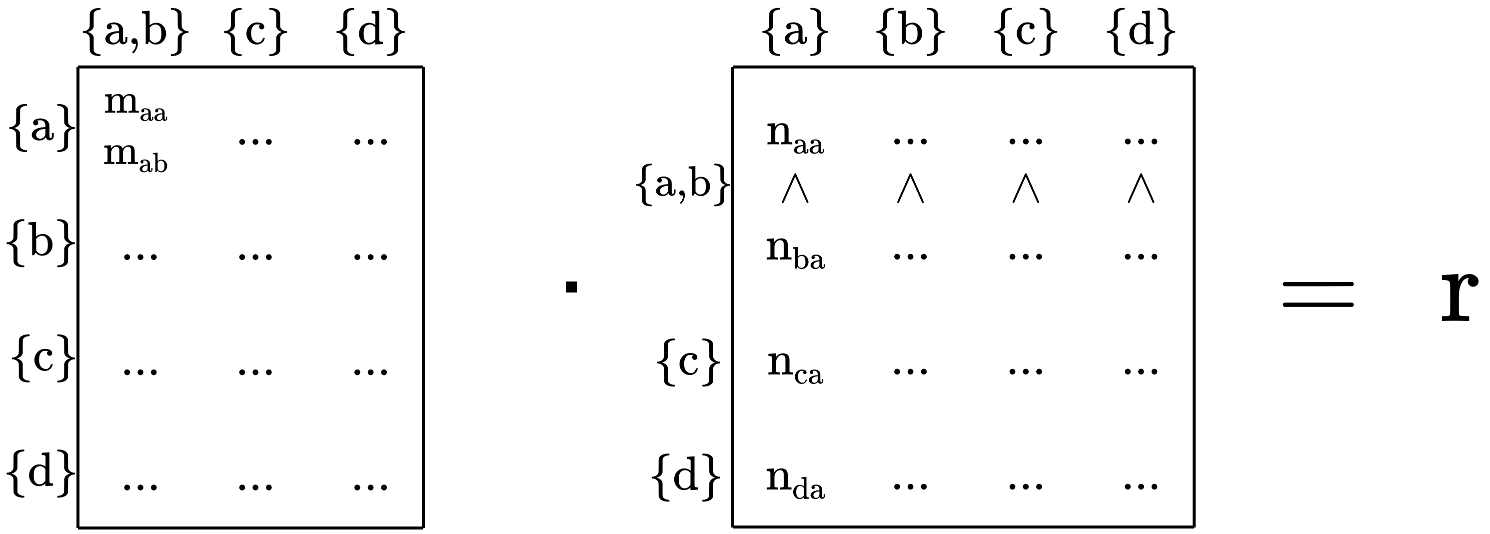
$\cdot$

	a	b	c	d
{a,b}	$n_{aa} \wedge n_{ba}$	... $\wedge$ ...	... $\wedge$ ...	... $\wedge$ ...
{c}	n_{ca}	...	...	...
{d}	n_{da}	...	...	...

$= \mathrm{r}$

$$r_{aa} = (m_{aa} + n_{aa} \wedge n_{ba}) \wedge (m_{ac} + n_{ca}) \wedge (m_{ad} + n_{da})$$

Compacting rows



On average, compaction of rows and columns reduces space consumption of matrices by 5 orders of magnitude!

In practice…

In practice...

- Grammars are **recursive** objects.
  We need to solve **mutually recursive** and monotonous equations**.**

In practice…

- Grammars are **recursive** objects.
  We need to solve **mutually recursive** and monotonous equations**.**
- Lots of **partitions** are computed: efficient data representations (bit sets) and algorithms are needed

In practice...

- Grammars are **recursive** objects.
  We need to solve **mutually recursive** and monotonous equations**.**
- Lots of **partitions** are computed: efficient data representations (bit sets) and algorithms are needed
- Re-ordering of matrix product chains
- Maximal sharing of intermediate matrices
- ...

In practice...

- Grammars are **recursive** objects.
  We need to solve **mutually recursive** and monotonous equations**.**
- Lots of **partitions** are computed: efficient data representations (bit sets) and algorithms are needed
- Re-ordering of matrix product chains
- Maximal sharing of intermediate matrices
- ...

You will find these explained in our paper!

Conclusion

Conclusion

A new algorithm that provides a significant speed-up to LR(1) reachability by:

- Reframing the problem. Solving a set of **mutually recursive** equations on **matrices**.
- **Compacting** matrices in a sound way.

The implementation gives good results and is available in the current release of Menhir.