From Traces To Proofs: Proving Concurrent Programs Safe

S. Arun-Kumar

(Joint work with Chinmay Narayan, Subodh Sharma, and Shibashis Guha)

Indian Institute of Technology Delhi

Overview

Motivation

Example: Peterson's Algorithm

Example trace

$flag_1 = \text{true}, flag_2 = \text{true}, turn = 0$

while(true) **do**	**while**(true) **do**
a. $flag_1 := \text{true}$	p. $flag_2 := \text{true}$
b. $turn := 2$	q. $turn := 1$
A. **assume**($\neg flag_2 \parallel turn = 1$);	P. **assume**($\neg flag_1 \parallel turn = 2$);
CS1. $cs := 1$;	CS2. $cs := 2$;
d. $\ell_1 := cs$;	s. $\ell_2 := cs$;
e. $flag_1 := \text{false}$	t. $flag_2 := \text{false}$
od	**od**

Informal reasoning about the correctness of the mutual exclusion property

Example: Peterson's Algorithm

Example trace

$flag_1 = \text{true}, flag_2 = \text{true}, turn = 0$

while(true) **do**	**while**(true) **do**
a. $flag_1 := \text{true}$	p. $flag_2 := \text{true}$
b. $turn := 2$	q. $turn := 1$
A. **assume**($\neg flag_2 \parallel turn = 1$);	P. **assume**($\neg flag_1 \parallel turn = 2$);
CS1. $cs := 1$;	CS2. $cs := 2$;
d. $\ell_1 := cs$;	s. $\ell_2 := cs$;
e. $flag_1 := \text{false}$	t. $flag_2 := \text{false}$
od	**od**

Informal reasoning about the correctness of the mutual exclusion property

- When $turn := 2$ is the last write to variable $turn$

Example: Peterson's Algorithm

Example trace

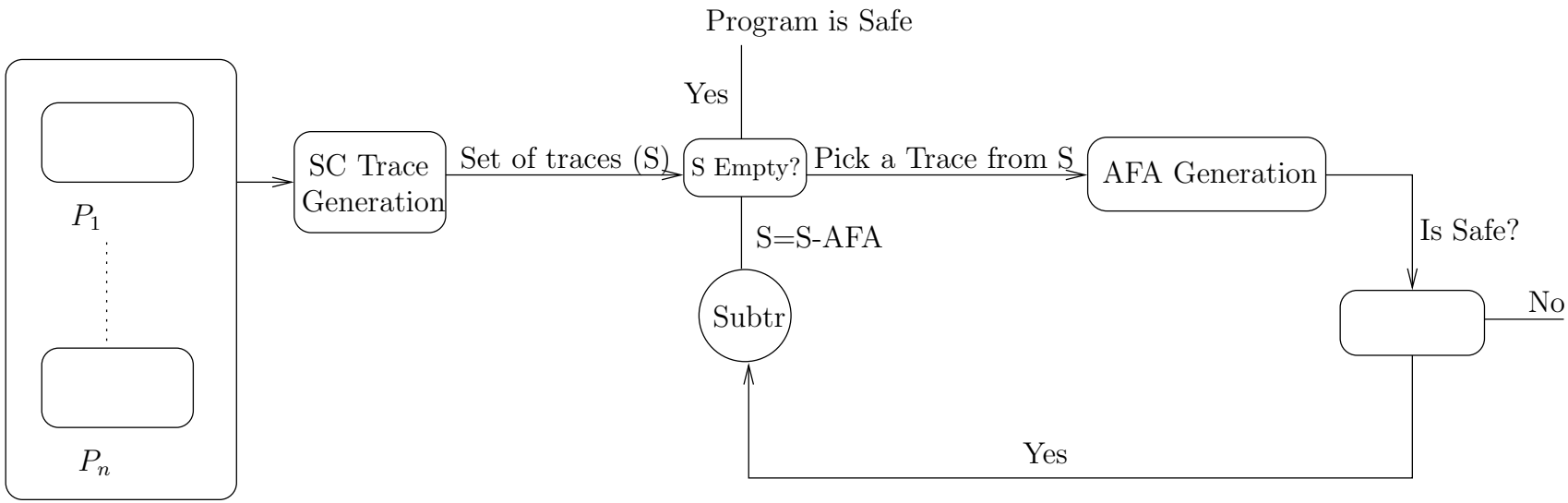
$flag_1 = \text{true}, flag_2 = \text{true}, turn = 0$

while(true) **do**	**while**(true) **do**
a. $flag_1 := \text{true}$	p. $flag_2 := \text{true}$
b. $turn := 2$	q. $turn := 1$
A. **assume**($\neg flag_2 \parallel turn = 1$);	P. **assume**($\neg flag_1 \parallel turn = 2$);
CS1. $cs := 1$;	CS2. $cs := 2$;
d. $\ell_1 := cs$;	s. $\ell_2 := cs$;
e. $flag_1 := \text{false}$	t. $flag_2 := \text{false}$
od	**od**

Informal reasoning about the correctness of the mutual exclusion property

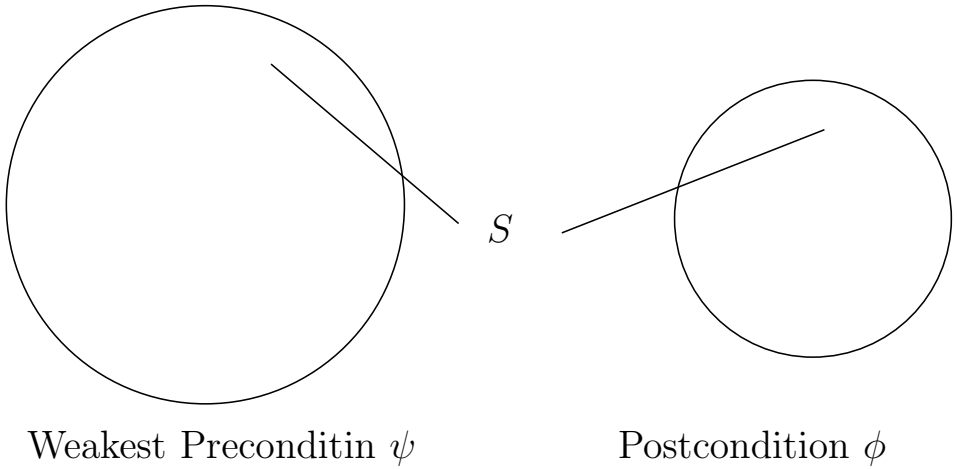
- When $turn := 2$ is the last write to variable $turn$
- When $turn := 1$ is the last write to variable $turn$

Overall Partitioning Algorithm



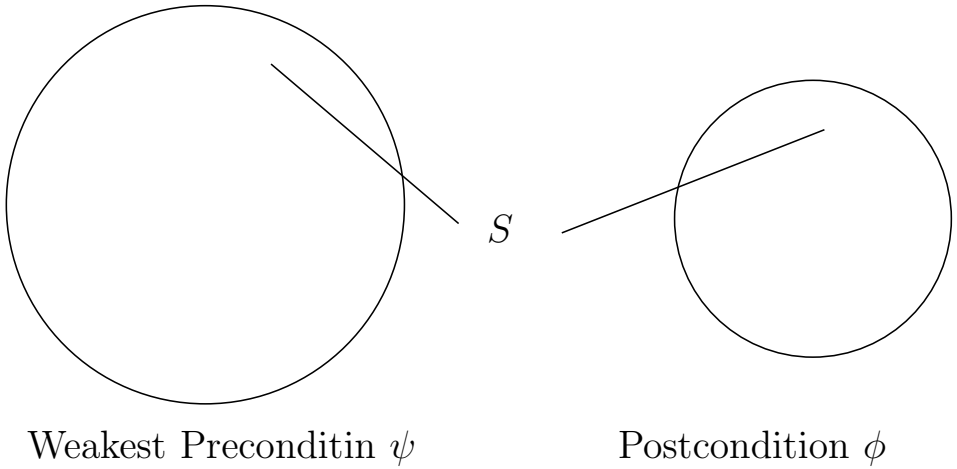
Some Preliminaries: Weakest Preconditions

$\mathtt{wp}(\mathtt{S}, \phi)$ is the largest set of states from where the execution of the statement S halts in a state satisfying ϕ.



Some Preliminaries: Weakest Preconditions

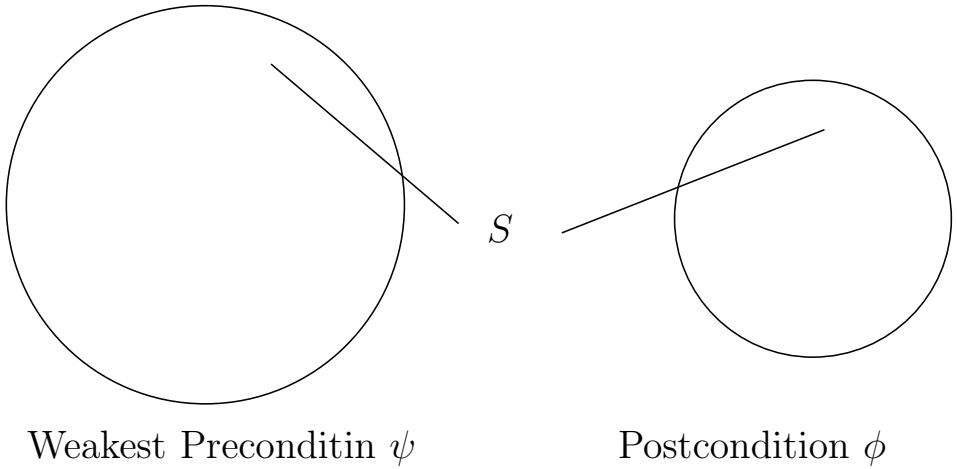
$\mathtt{wp}(\mathtt{S}, \phi)$ is the largest set of states from where the execution of the statement S halts in a state satisfying ϕ.



- $\mathtt{wp}(\mathtt{x} := \mathtt{a} + 1, \mathtt{x} > 5) = a + 1 > 5$

Some Preliminaries: Weakest Preconditions

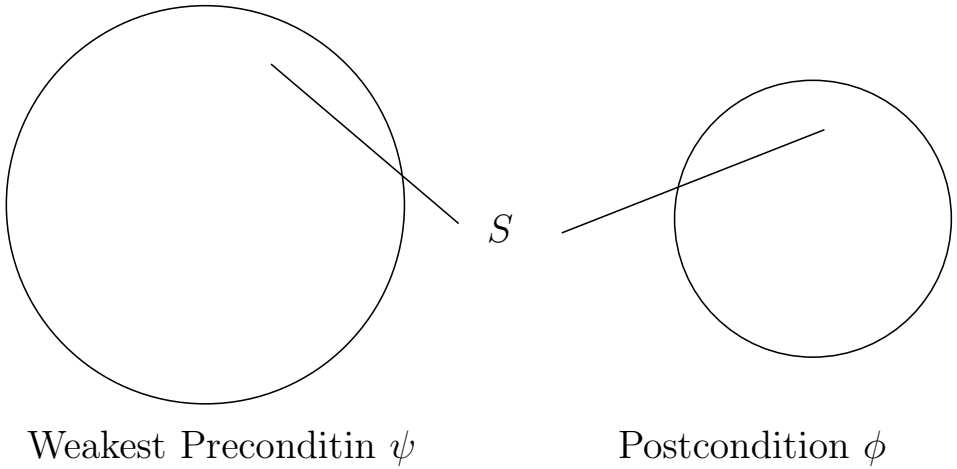
$\mathtt{wp}(\mathtt{S}, \phi)$ is the largest set of states from where the execution of the statement S halts in a state satisfying ϕ.



- $\mathtt{wp}(\mathtt{x} := \mathtt{a} + 1, \mathtt{x} > 5) = a + 1 > 5$
- $\mathtt{wp}(\mathtt{assert}(\psi), \phi) = \psi \ \wedge \ \phi$

Some Preliminaries: Weakest Preconditions

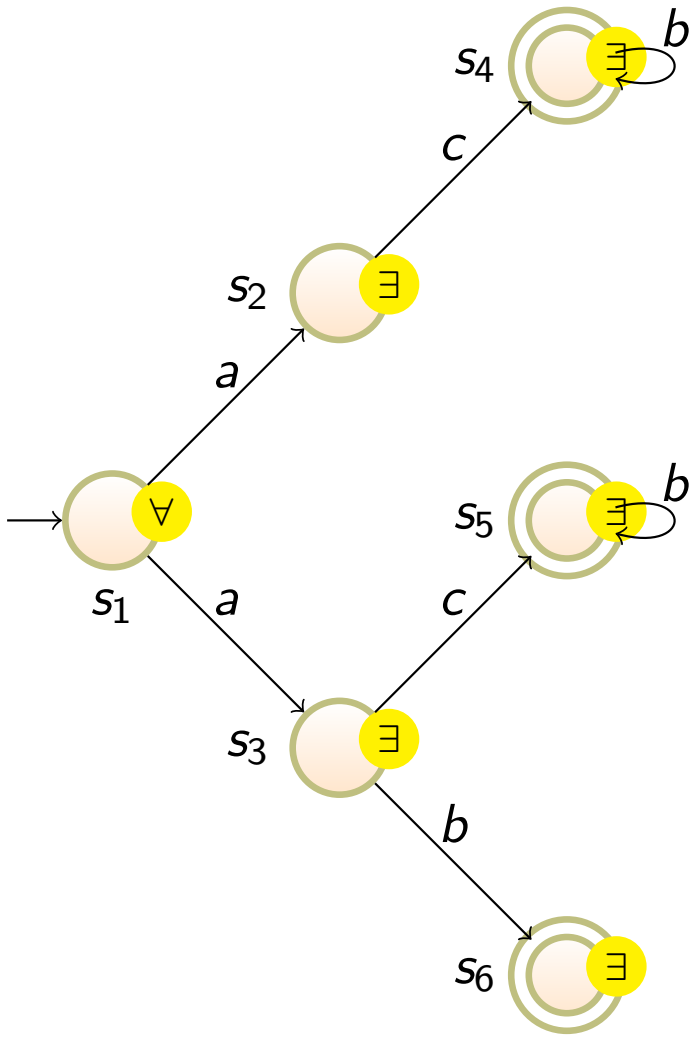
$\mathtt{wp}(\mathtt{S}, \phi)$ is the largest set of states from where the execution of the statement S halts in a state satisfying ϕ.



- $\mathtt{wp}(\mathtt{x} := \mathtt{a} + 1, \mathtt{x} > 5) = a + 1 > 5$
- $\mathtt{wp}(\mathtt{assert}(\psi), \phi) = \psi \ \wedge \ \phi$
- $\mathtt{wp}(\mathtt{assume}(\psi), \phi) = \psi \ \wedge \ \phi$

Preliminaries

Some Preliminaries: Alternating Finite Automata



- $\text{lang}(s_2) = cb^*$
- $\text{lang}(s_3) = cb^* + b$
- $\text{lang}(s_1) = acb^*$

Example

Example trace and its AFA construction

▸ Peterson

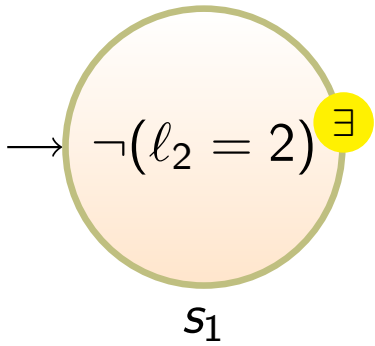
Example trace σ

$$
\begin{array}{ll}
\text{a.} & \mathit{flag}_1 := \text{true} \\
\text{b.} & \mathit{turn} := 2 \\
\text{A.} & \texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1); \\
\text{p.} & \mathit{flag}_2 := \text{true} \\
\text{q.} & \mathit{turn} := 1 \\
\text{P.} & \texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2); \\
\text{CS2.} & \mathit{cs} := 2 \\
\text{CS1.} & \mathit{cs} := 1 \\
\text{s.} & \ell_2 := \mathit{cs} \\
 & \text{assert}(\ell_2 = 2)
\end{array}
$$

Let $\mathcal{I}$ be an initial condition. Then show that $\texttt{wp}(\sigma, \neg(\ell_2 = 2)) \wedge \mathcal{I}$ is unsatisfiable

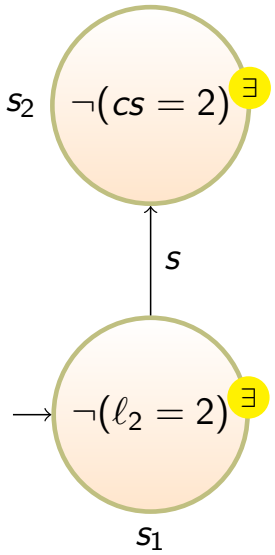
Step 1

a. $flag_1 := \mathsf{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \mathrm{flag}_2 \parallel \mathrm{turn} = 1);$
p. $flag_2 := \mathsf{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \mathrm{flag}_1 \parallel \mathrm{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\mathrm{assert}(\ell_2 = 2)$



Step 2

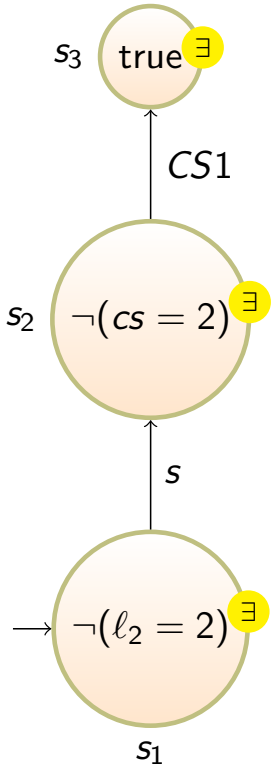
a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



Note that $\texttt{wp}(\texttt{s}, \neg(\ell_2 = 2)) = \neg(cs = 2)$

Step 3

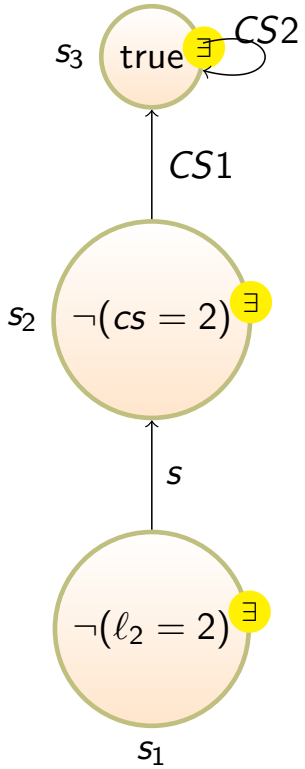
a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



Note that $\texttt{wp}(\texttt{c}, \neg(\texttt{cs} = 2)) = \text{true}$

Step 4

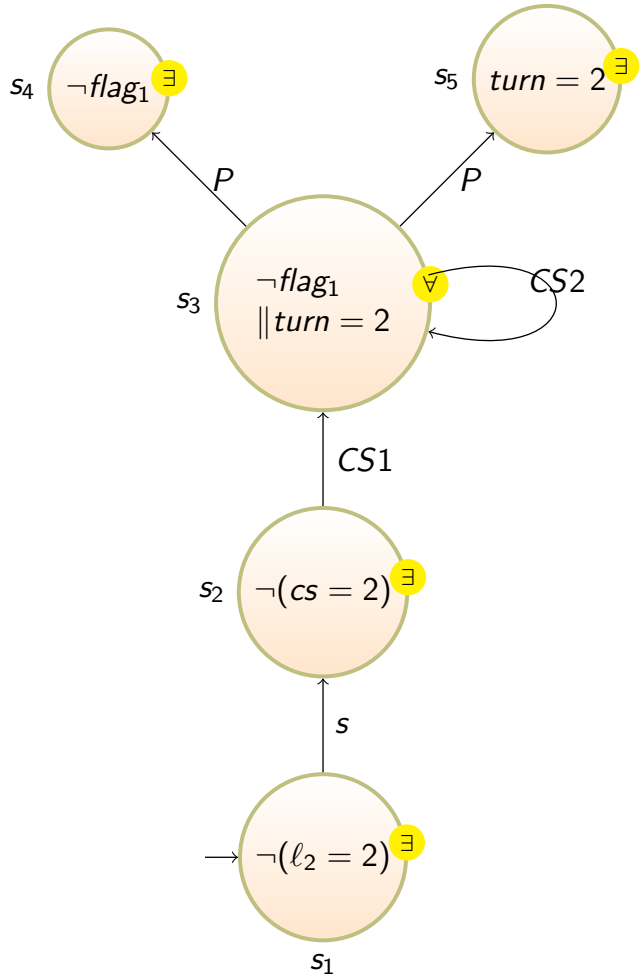
a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



Note that $\texttt{wp}(\texttt{CS2}, \text{true}) = \text{true}$

Step 5

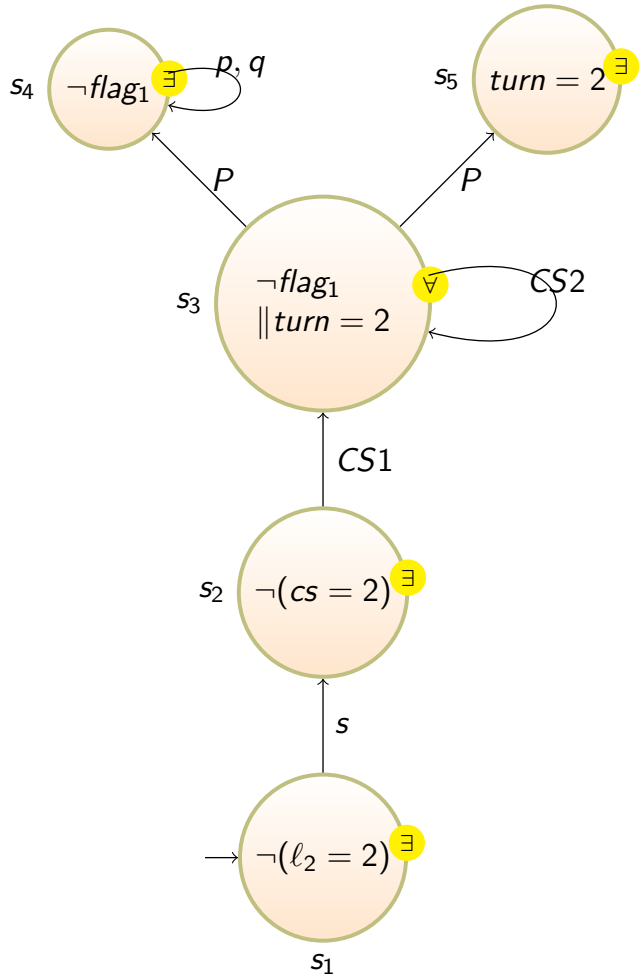
a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



Note that $\texttt{wp}(\texttt{P}, \text{true}) = (flag_1 = 0 \| turn = 2)$

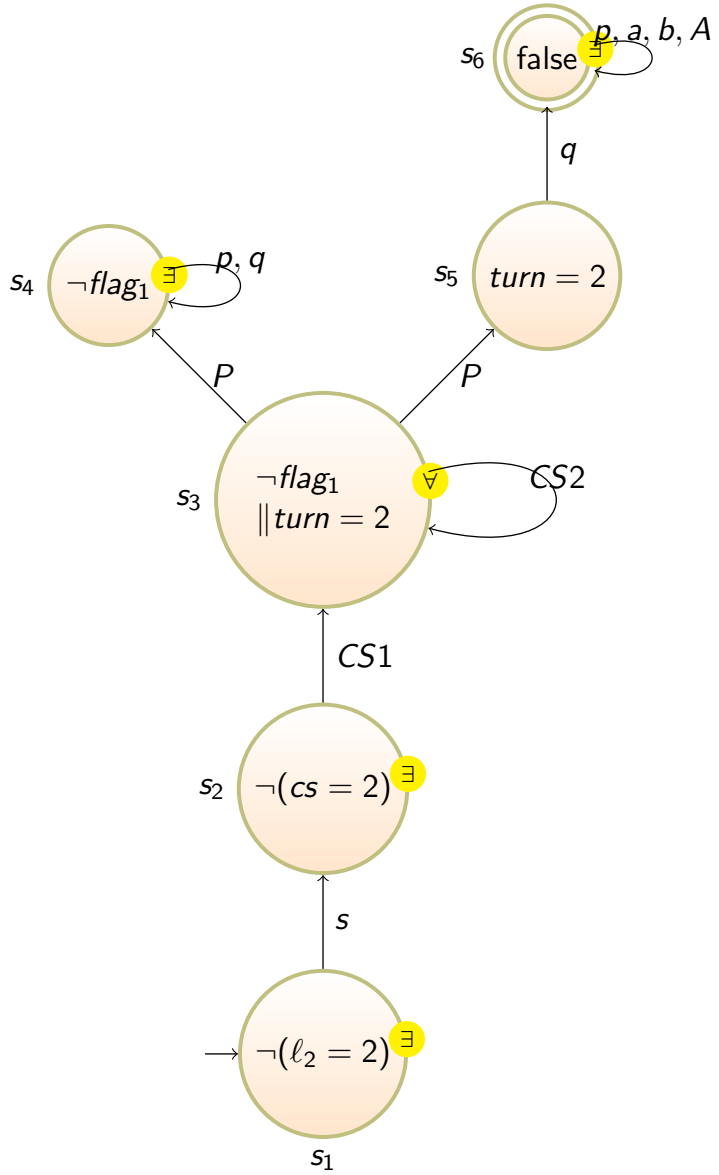
Step 6

a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



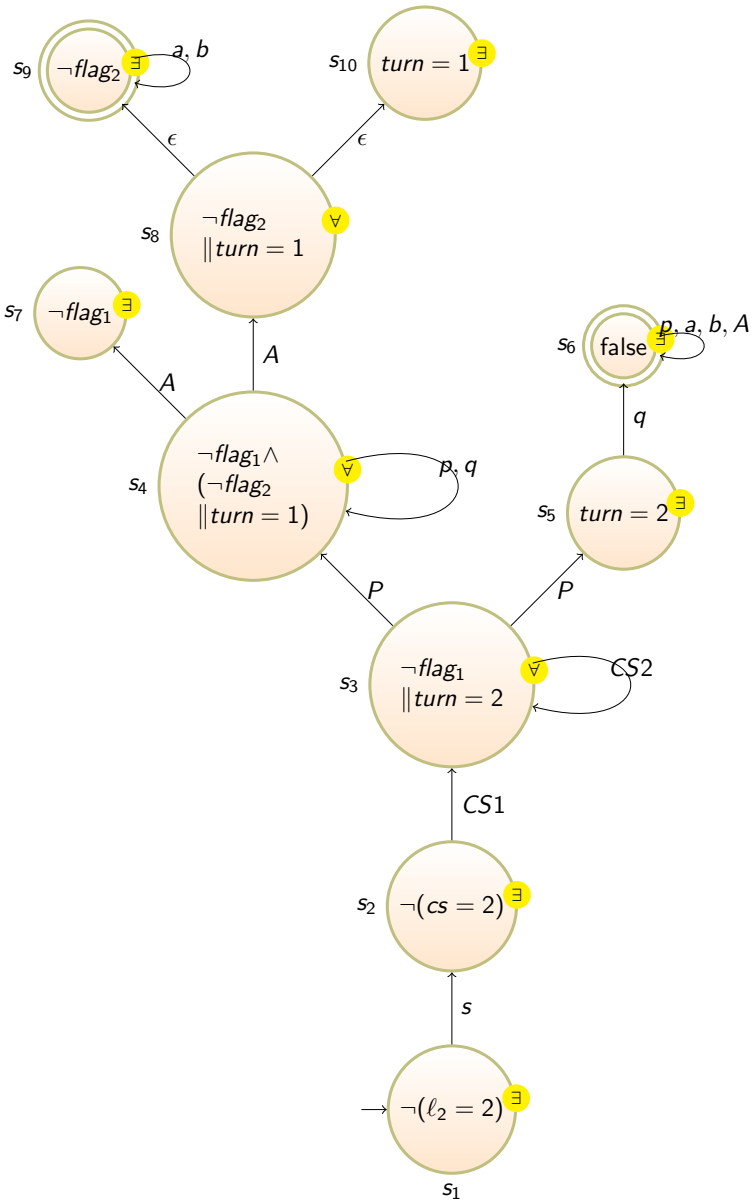
Step 7

a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



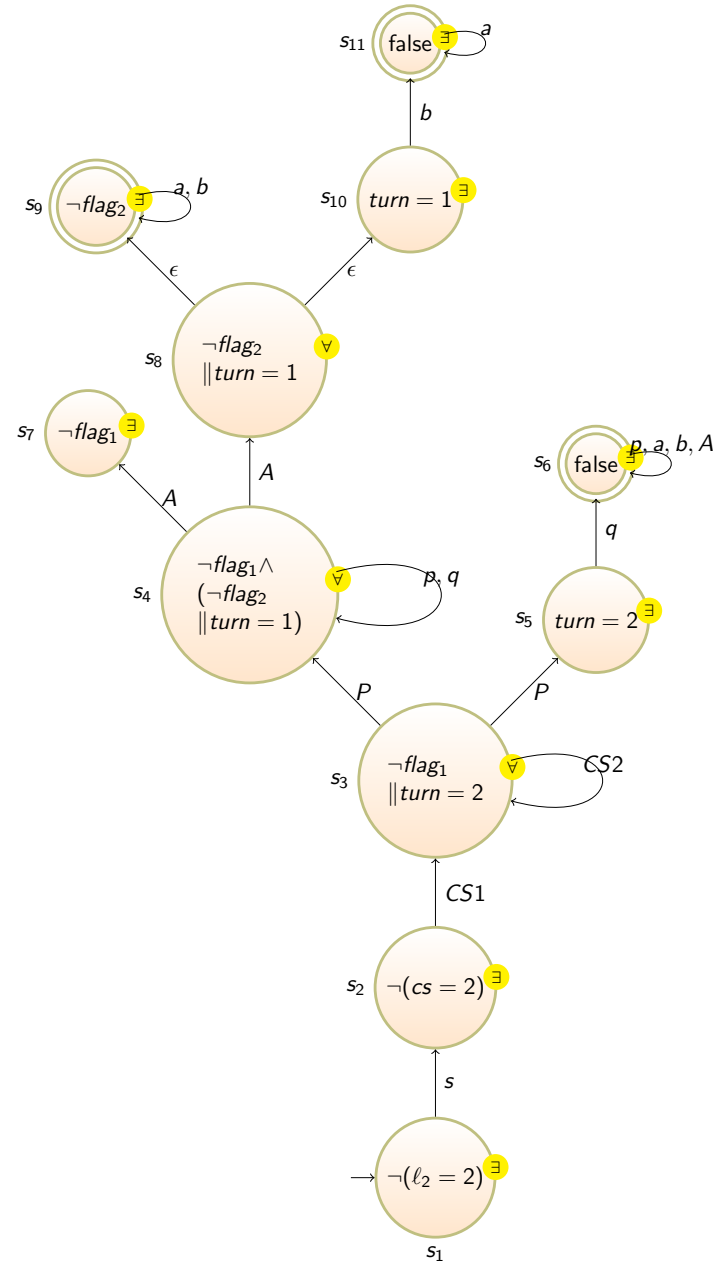
Step 8

a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



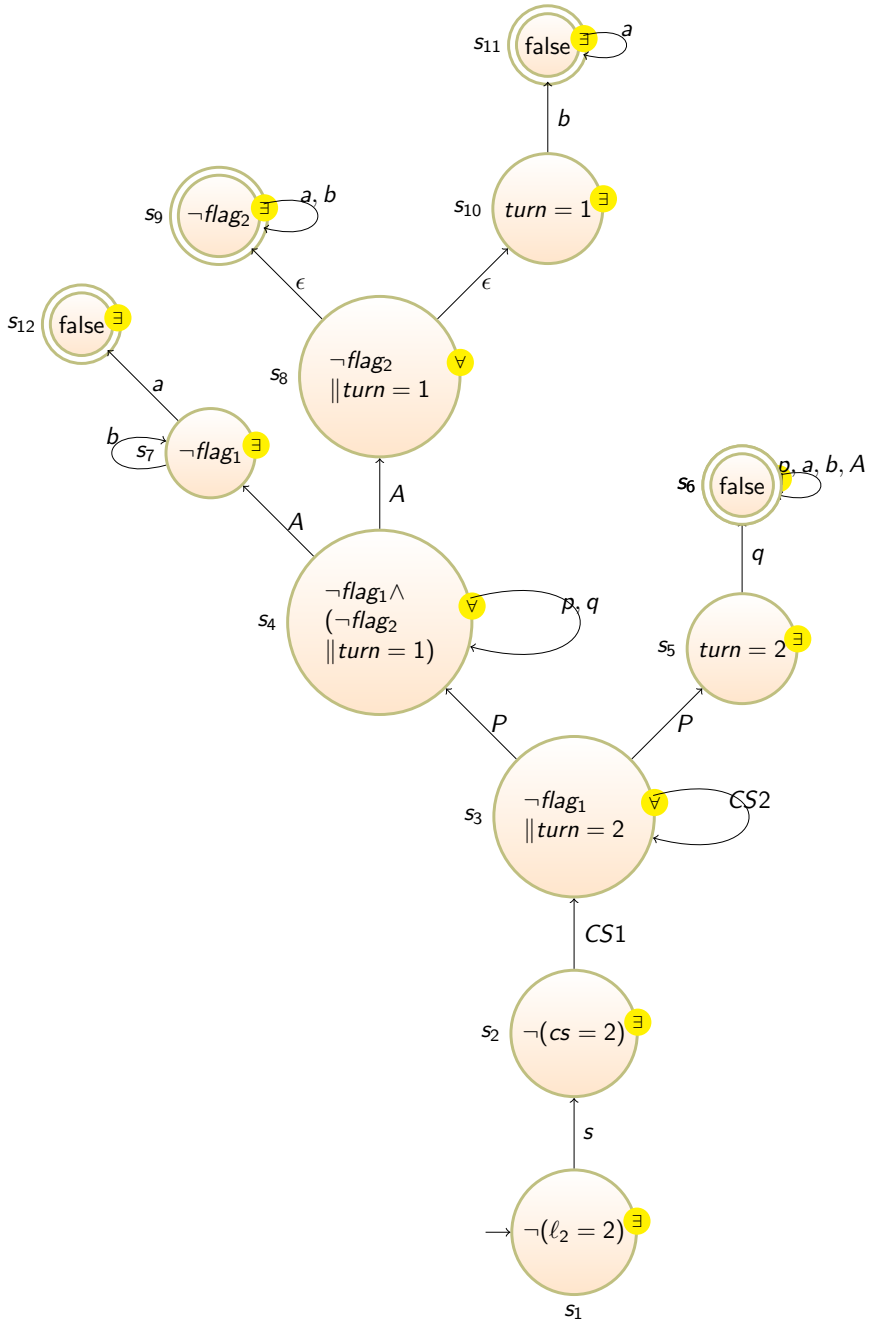
Step 9

a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$

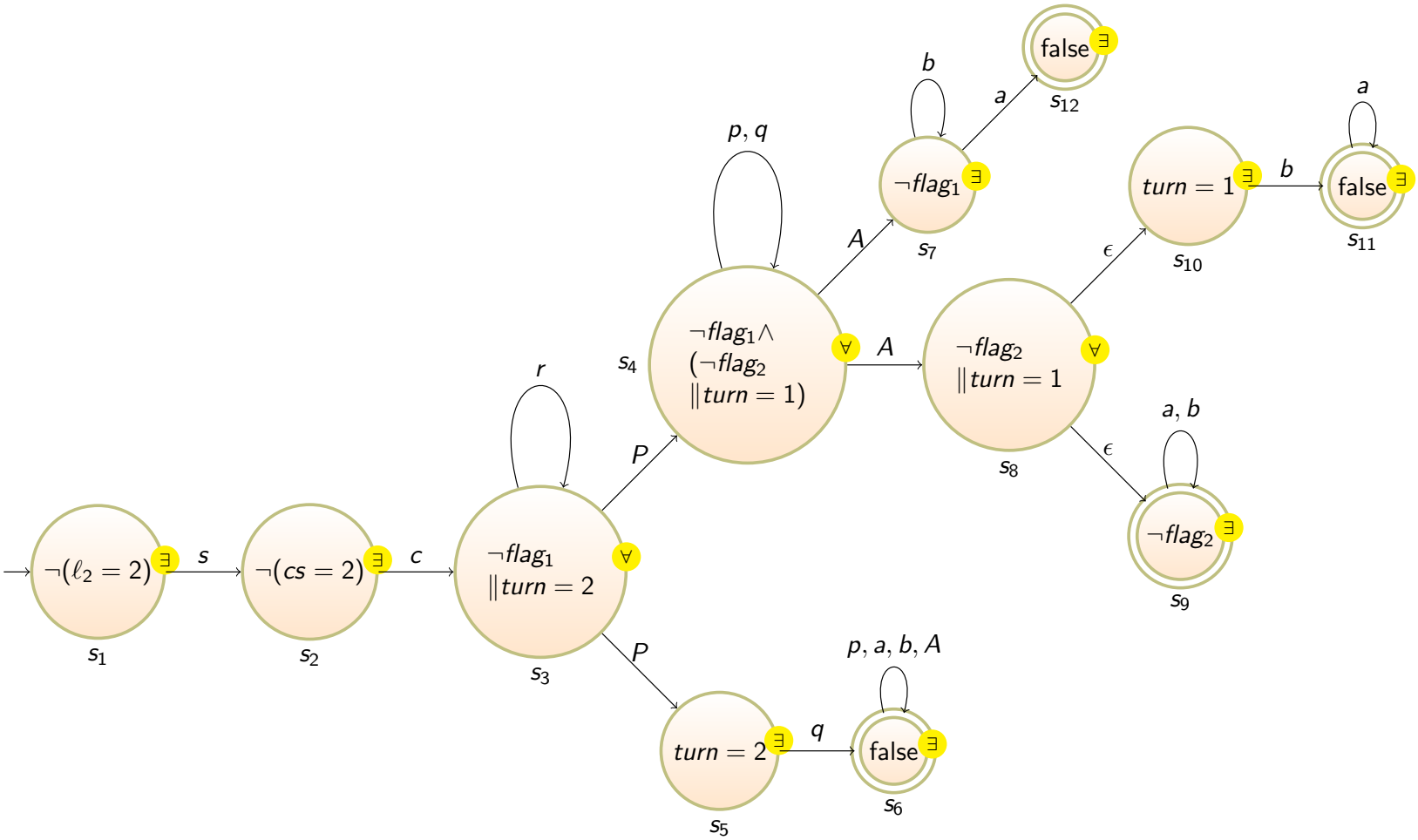


Step 10

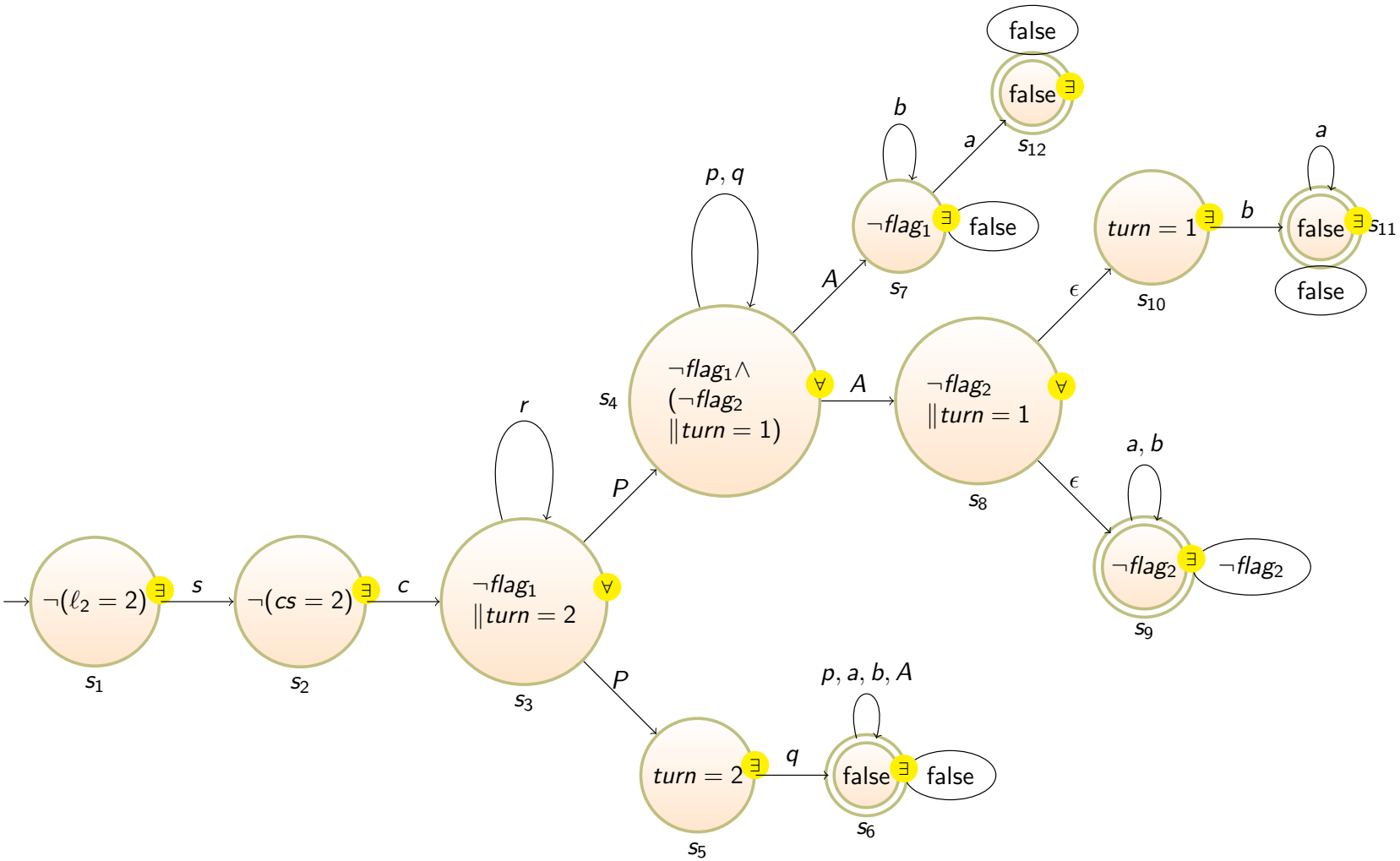
a. $flag_1 := \text{true}$
b. $turn := 2$
A. $\texttt{assume}(\neg \text{flag}_2 \parallel \text{turn} = 1);$
p. $flag_2 := \text{true}$
q. $turn := 1$
P. $\texttt{assume}(\neg \text{flag}_1 \parallel \text{turn} = 2);$
CS2. $cs := 2$
CS1. $cs := 1$
s. $\ell_2 := cs$
$\text{assert}(\ell_2 = 2)$



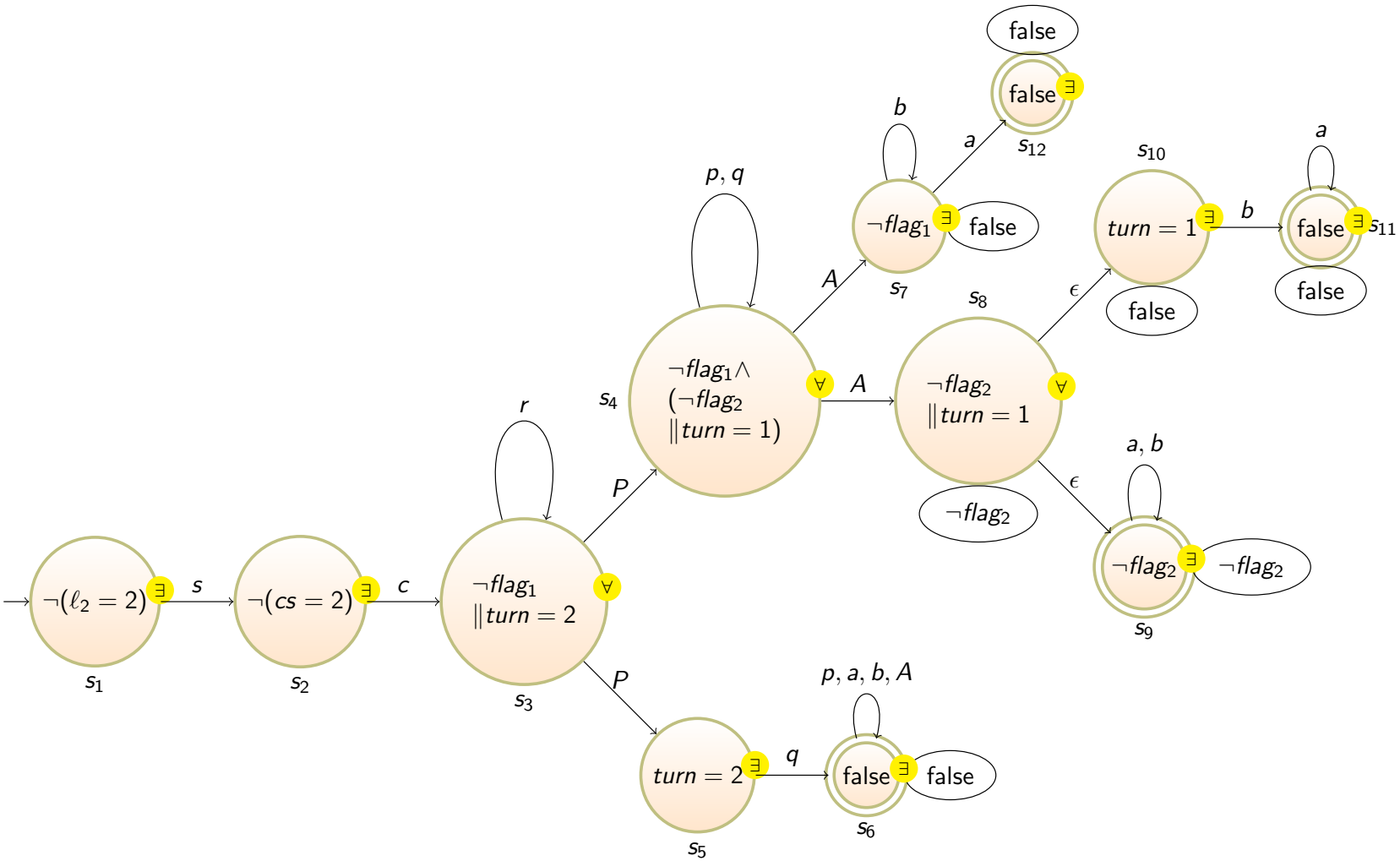
AFA constructed from the given trace



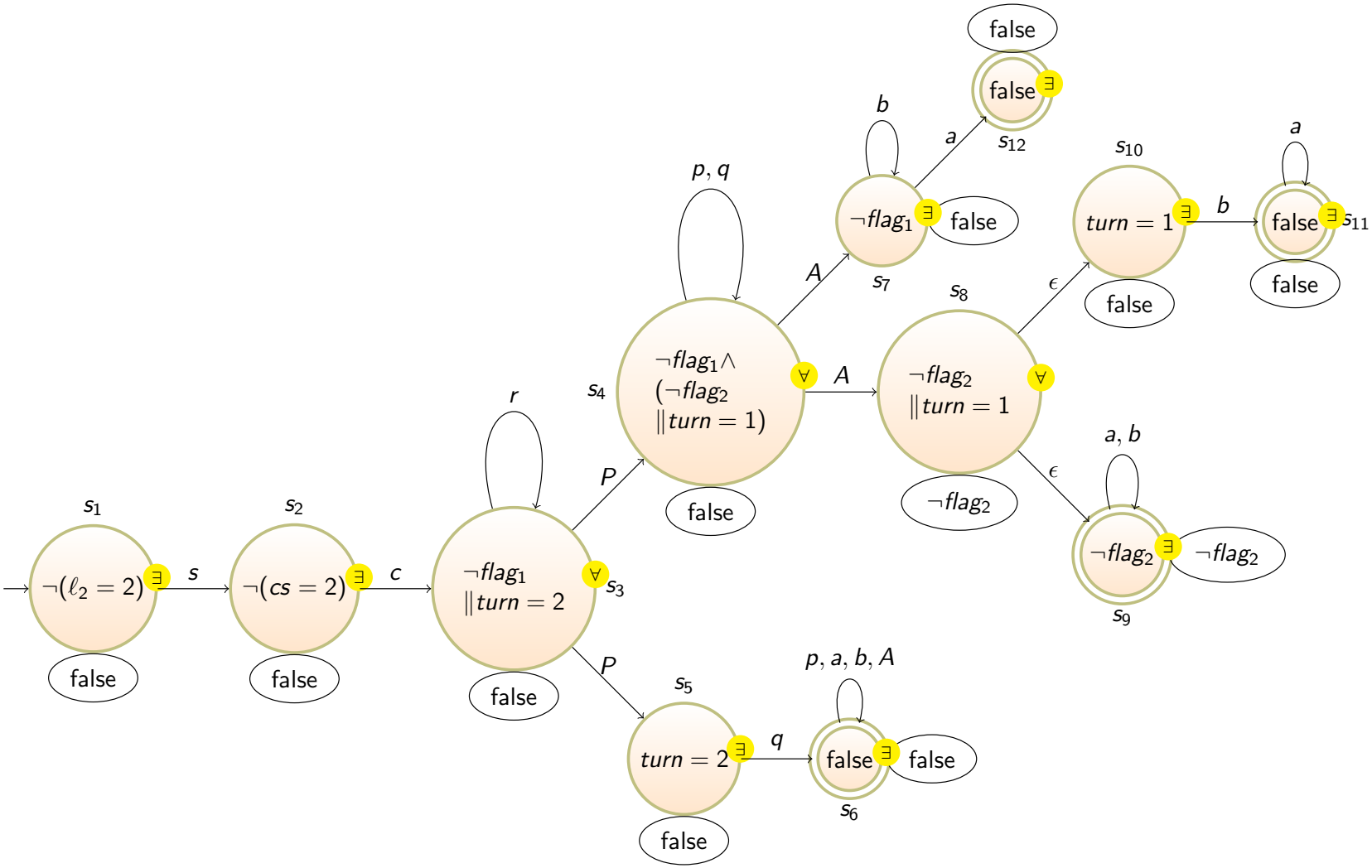
Weakest precondition generation from this AFA



Weakest precondition generation from this AFA



Weakest precondition generation from this AFA: Final



Overall Picture

Overall Picture

- Let $\mathcal{A}_{\sigma,\neg\phi}$ be the AFA constructed from the trace σ and the safety assertion $\neg\phi$
- Let ψ be the annotation that flows to the initial state then

1. For every word σ' accepted by the AFA $\mathcal{A}_{\sigma,\neg\phi}$

> **Theorem 1:** $\texttt{wp}(\texttt{rev}(\sigma'), \neg\phi)$ is same as ψ

2. $rev(\sigma)$ is also accepted by the AFA $\mathcal{A}_{\sigma,\neg\phi}$
3. If $\psi \wedge \mathcal{I}$ is unsatisfiable then Safe to exclude the words of this automaton from safety checking

Overall Picture

- Let $\mathcal{A}_{\sigma,\neg\phi}$ be the AFA constructed from the trace σ and the safety assertion $\neg\phi$
- Let ψ be the annotation that flows to the initial state then

1. For every word σ' accepted by the AFA $\mathcal{A}_{\sigma,\neg\phi}$

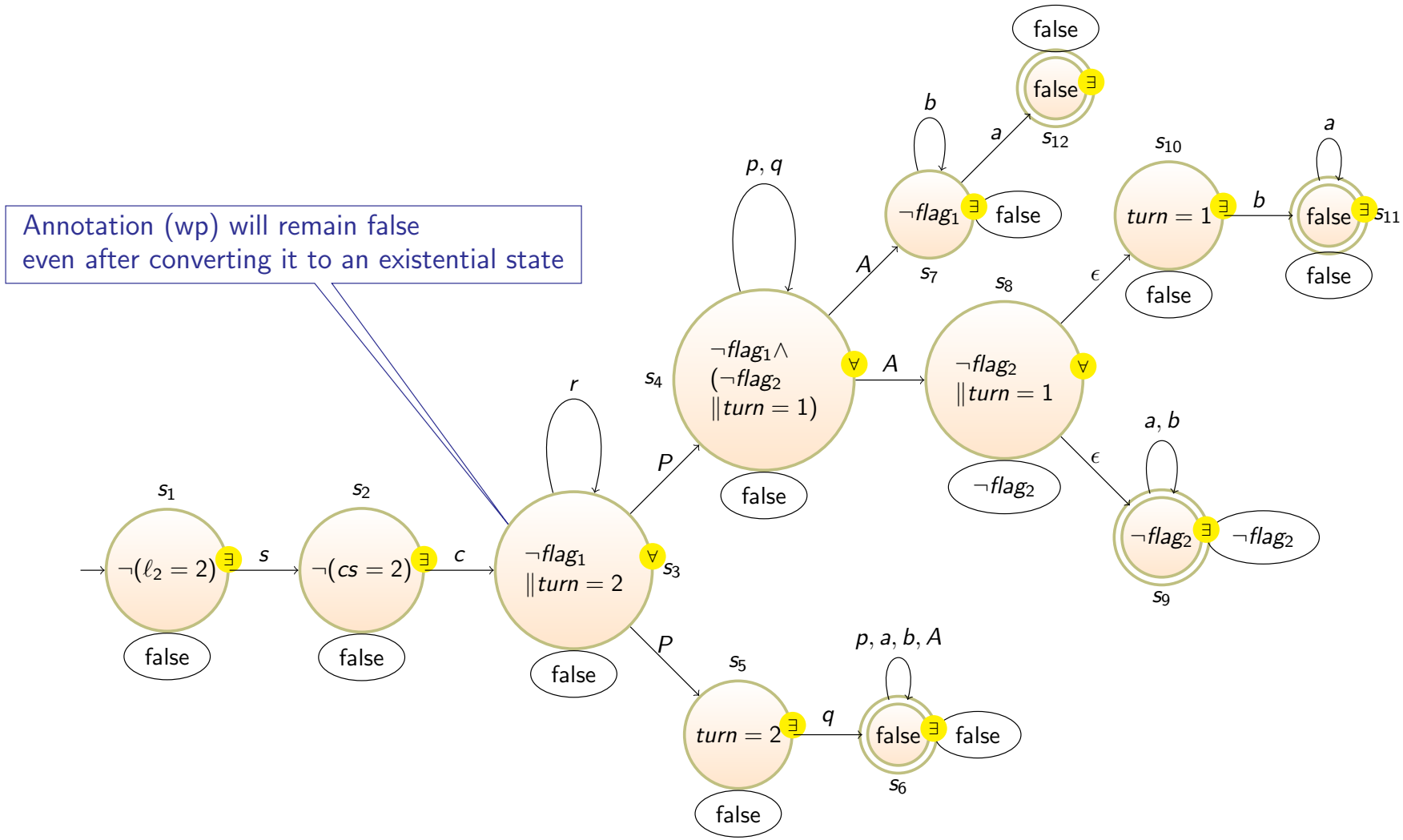
> **Theorem 1:** $\texttt{wp}(\texttt{rev}(\sigma'), \neg\phi)$ is same as ψ

2. $rev(\sigma)$ is also accepted by the AFA $\mathcal{A}_{\sigma,\neg\phi}$
3. If $\psi \wedge \mathcal{I}$ is unsatisfiable then Safe to exclude the words of this automaton from safety checking

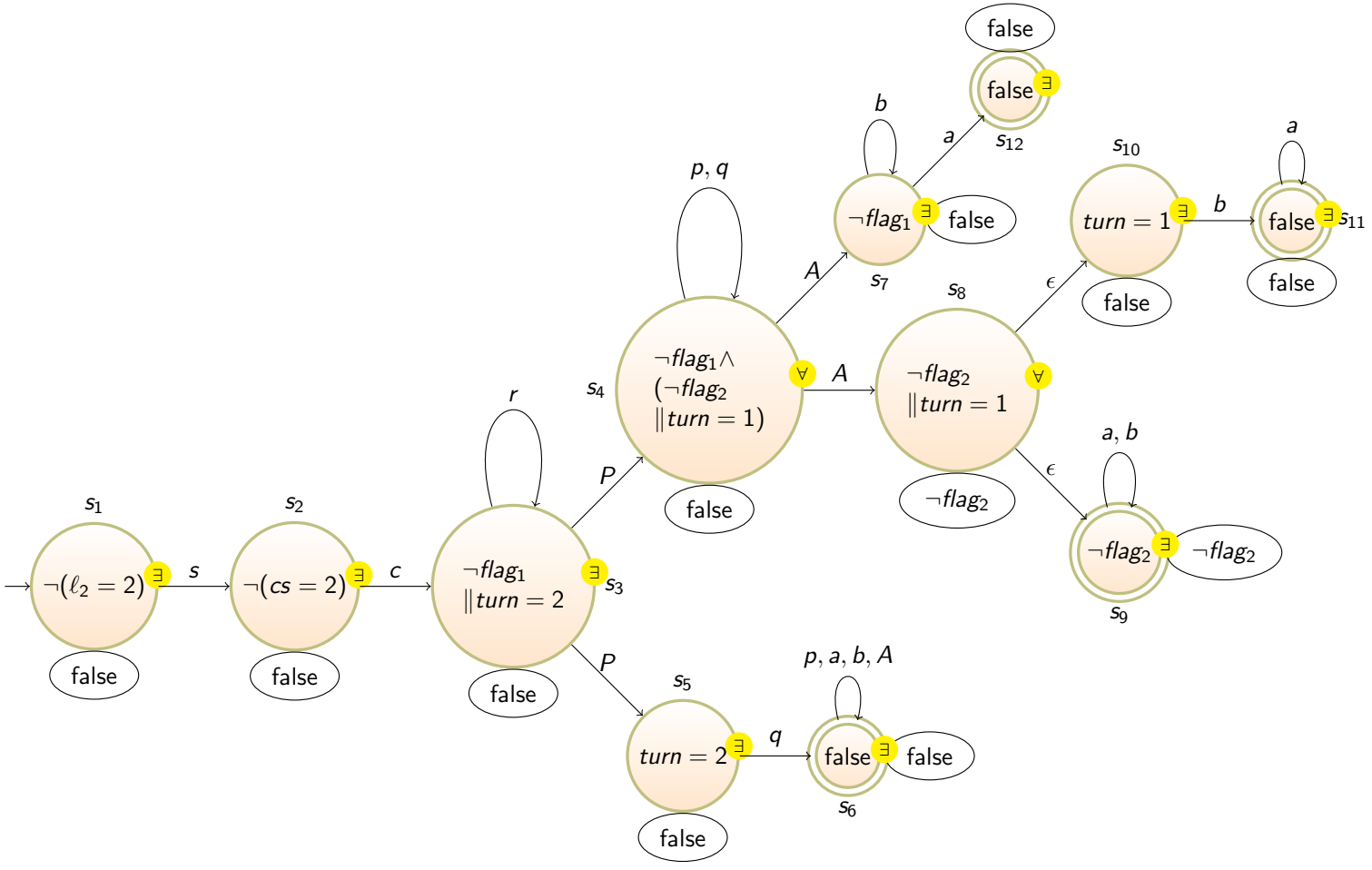
Optimizations to increase the set of accepted words but preserving Theorem 1

Optimizations

Optimization 1: Conversion from Universal to Existential states

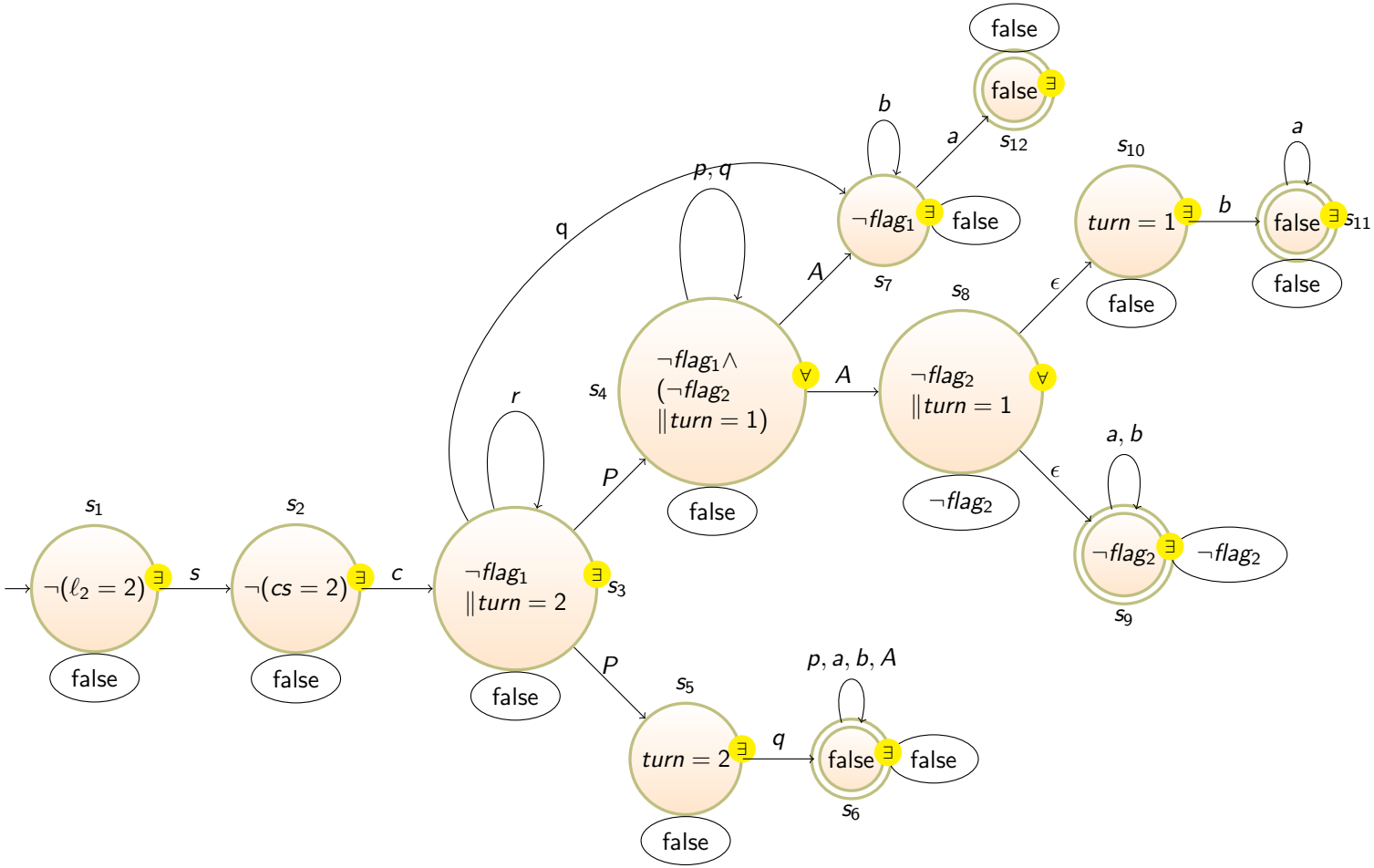


Optimization 2: Adding Extra Edges



Label q in Peterson's algorithm is $turn := 1$

Optimization 2: Adding Extra Edges



Label q in Peterson's algorithm is $turn := 1$

Performance Evaluation

Performance Evaluation- Time in Seconds

Program	ProofTraPar	THREADER[GPR11]	Lazy-CSeq[ITF+14]
Peterson.safe	**0.3**	3.2	3.1
Dekker.safe	**1.1**	1.7	4.2
Lamport.safe	**2.4**	47	5.1
Szymanksi.safe	**3**	12.8	4
TimeVarMutex.safe	**0.76**	8.56	4.2
RWLock.safe (2R+2W)	8.8	140	**6.7**
RWLock.unsafe (2R+2W)	3.8	153	**0.7**
Qrcu.safe (2R+1W)	**20**	–	41
Qrcu.unsafe (2R+1W)	13.8	76	**1.1**

Contribution

Contribution

- A novel algorithm to directly construct an AFA to give a sound and complete verification algorithm

- Demonstrated the feasibility of trace partitioning approach by implementing and comparing it against state-of-the-art tools

Remarks

Remarks

1. Trace partitioning and Partial Order Reduction

2. Comparison to *inductive Data Flow graph*(iDFG) proposed in Farzan et.al [FKP13]

    - *iDFG* Converted to AFA for set theoretic operations

    - No implementation hence difficult to compare the cost of direct AFA construction vs iDFG to AFA construction

Thank You


A. Farzan, Z. Kincaid, and A. Podelski.
Inductive data flow graphs.
In *POPL*, pages 129–142, 2013.


A. Gupta, C. Popeea, and A. Rybalchenko.
Threader: A constraint-based verifier for multi-threaded programs.
In *CAV*, pages 412–417, 2011.


O. Inverso, E. Tomasco, B. Fischer, S. La Torre, and G. Parlato.
Bounded model checking of multi-threaded C programs via lazy sequentialization.
In *CAV*, volume 8559 of *LNCS*, pages 585–602. Springer, 2014.