Causal Termination of Multi-threaded Programs

Andrey Kupriyanov and Bernd Finkbeiner

Saarland University
Reactive Systems Group

July 22, 2014

UNIVERSITÄT
DES
SAARLANDES

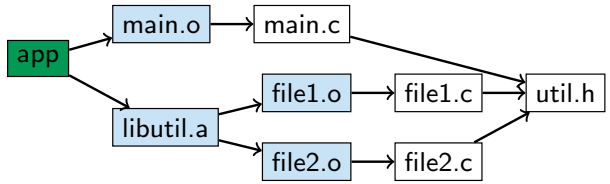
Reactive
Systems

Termination of multi-threaded programs

- Parallel compilation (e.g. GNU Make)

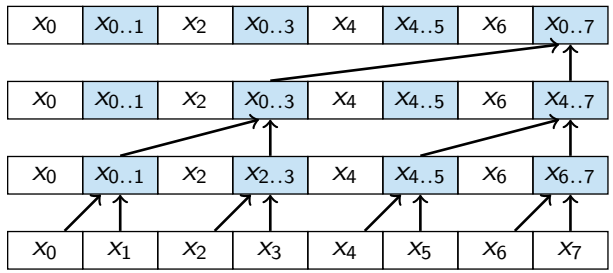
  gnu.org/software/make/

  ```
  make -j N
  ```



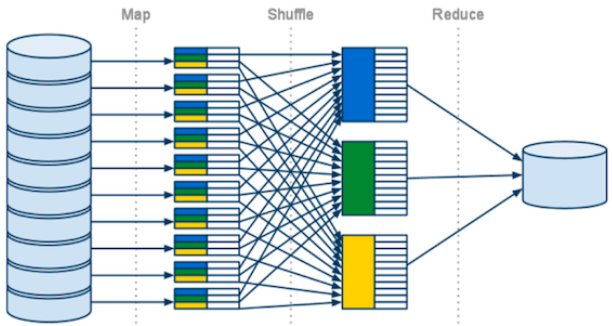
- Parallel computations in GPUs (OpenCL, CUDA)

  developer.nvidia.com/cuda-zone/



- Distributed processing (e.g. the Map-Reduce architecture)

  developers.google.com/appengine/docs/java/dataprocessing/



- Device drivers, leader election, ...

Producer-Consumer (Map-Reduce architecture)

Producer 1

```
while (p1>0) {
  if(*) q1++;
  else  q2++;
  p1--;
}
```

Producer 2

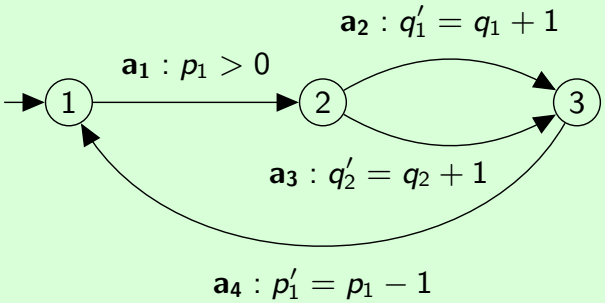
```
while (p2>0) {
  if(*) q1++;
  else  q2++;
  p2--;
}
```

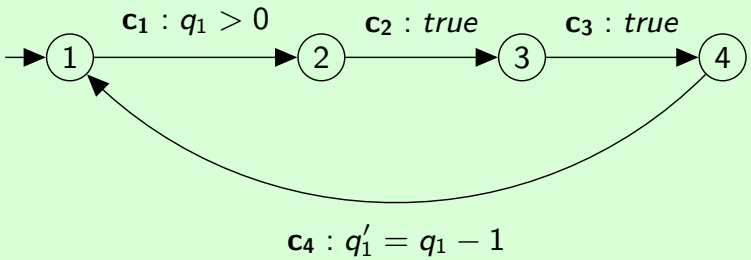
Consumer 1

```
while (true) {
  await(q1>0);
  skip; //step 1
  skip; //step 2
  q1--;
}
```

Consumer 2

```
while (true) {
  await(q2>0);
  skip; //step 1
  skip; //step 2
  q2--;
}
```





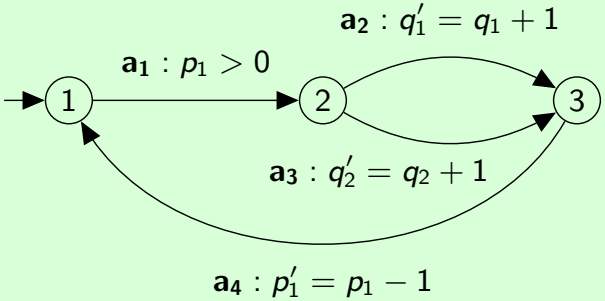
Producer-Consumer (Map-Reduce architecture)

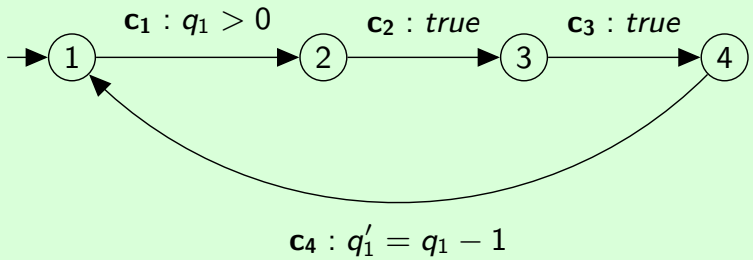
Producer 1
```
while (p1>0) {
  if(*) q1++;
  else  q2++;
  p1--;
}
```

Producer 2
```
while (p2>0) {
  if(*) q1++;
  else  q2++;
  p2--;
}
```

Consumer 1
```
while (true) {
  await(q1>0);
  skip; //step 1
  skip; //step 2
  q1--;
}
```

Consumer 2
```
while (true) {
  await(q2>0);
  skip; //step 1
  skip; //step 2
  q2--;
}
```





	Terminator		T2		AProVE	
Threads	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)
1	3.37	26	2.42	38	3.17	237
2	1397	1394	3.25	44	6.79	523
3	×	MO	U(29.2)	253	U(26.6)	1439
4	×	MO	U(36.6)	316	U(71.2)	1455
5	×	MO	U(30.7)	400	U(312)	1536
10	×	MO	Z3-TO	×	×	MO
20	×	MO	Z3-TO	×	×	MO
40	×	MO	Z3-TO	×	×	MO
60	×	MO	Z3-TO	×	×	MO
80	×	MO	Z3-TO	×	×	MO
100	×	MO	Z3-TO	×	×	MO

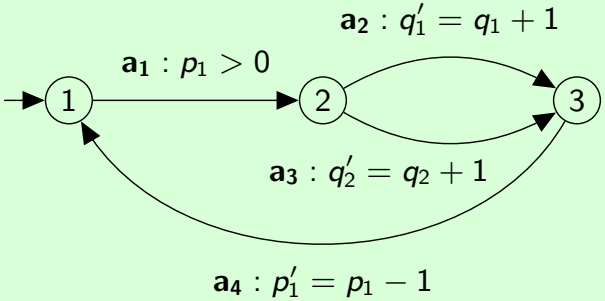
Producer-Consumer (Map-Reduce architecture)

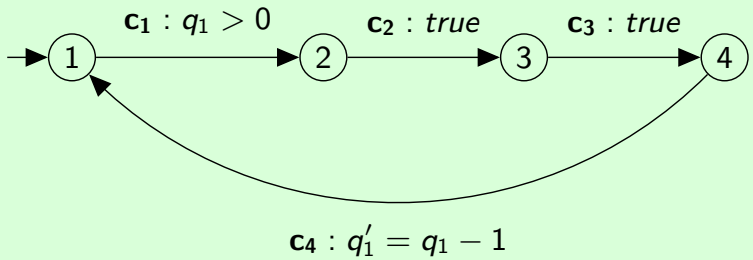
Producer 1
```
while (p1>0) {
  if(*) q1++;
  else  q2++;
  p1--;
}
```

Producer 2
```
while (p2>0) {
  if(*) q1++;
  else  q2++;
  p2--;
}
```

Consumer 1
```
while (true) {
  await(q1>0);
  skip; //step 1
  skip; //step 2
  q1--;
}
```

Consumer 2
```
while (true) {
  await(q2>0);
  skip; //step 1
  skip; //step 2
  q2--;
}
```





	Terminator		T2		AProVE		**Arctor**	
Threads	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)
1	3.37	26	2.42	38	3.17	237	**0.002**	**2.3**
2	1397	1394	3.25	44	6.79	523	**0.002**	**2.6**
3	×	MO	U(29.2)	253	U(26.6)	1439	**0.002**	**2.6**
4	×	MO	U(36.6)	316	U(71.2)	1455	**0.003**	**2.7**
5	×	MO	U(30.7)	400	U(312)	1536	**0.007**	**2.7**
10	×	MO	Z3-TO	×	×	MO	**0.027**	**3.0**
20	×	MO	Z3-TO	×	×	MO	**0.30**	**4.2**
40	×	MO	Z3-TO	×	×	MO	**4.30**	**12.7**
60	×	MO	Z3-TO	×	×	MO	**20.8**	**35**
80	×	MO	Z3-TO	×	×	MO	**67.7**	**145**
100	×	MO	Z3-TO	×	×	MO	**172**	**231**

Causality

In the spirit of the FLoC panel on Saturday...

- *why are human-created artifacts easy to analyze?*
- *can an automatic tool capture the intuition of artifact's creator?*

Karem A. Sakallah

Our view is that humans are good at:

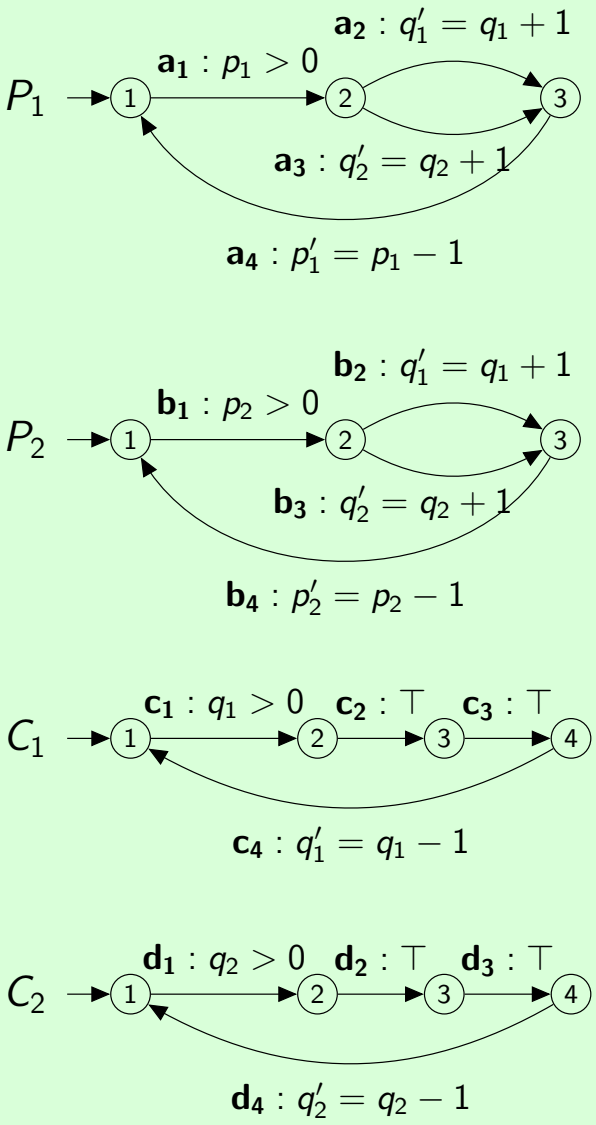
- tracking dependence / independence
- making (chains of) causal inferences
- recognizing (and reusing) similarities

Main problem for automation: standard state-based methods do not fully support these phenomena

Our approach

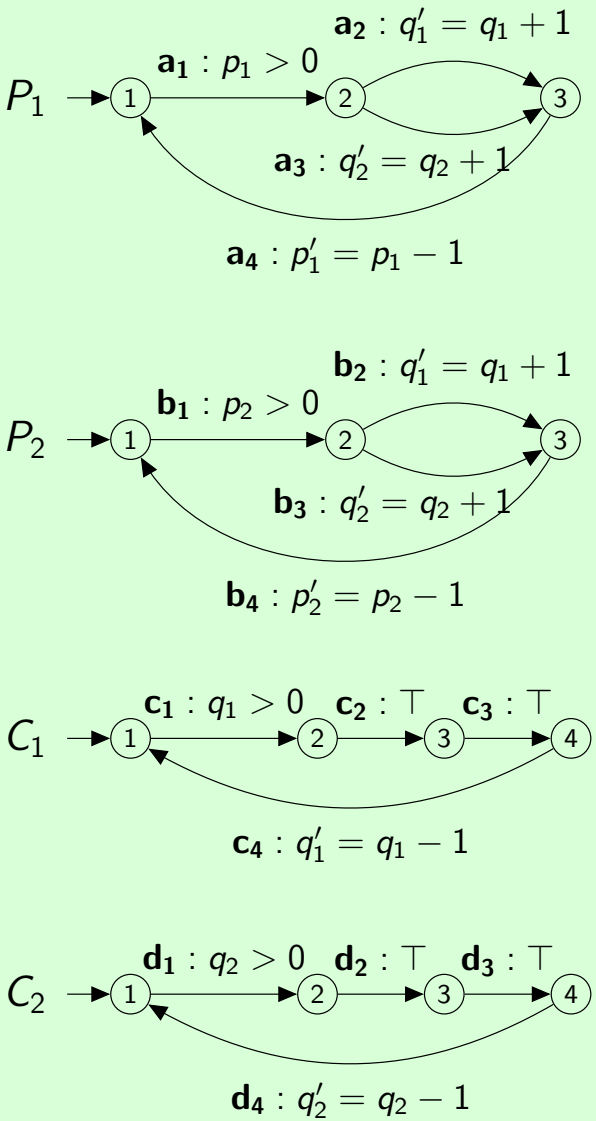
- dependence / independence $\Longrightarrow$ concurrent traces
- causal inferences $\Longrightarrow$ language-preserving trace transformations
- similarity recognition $\Longrightarrow$ tableau-based trace search

Producer-Consumer: a causal proof

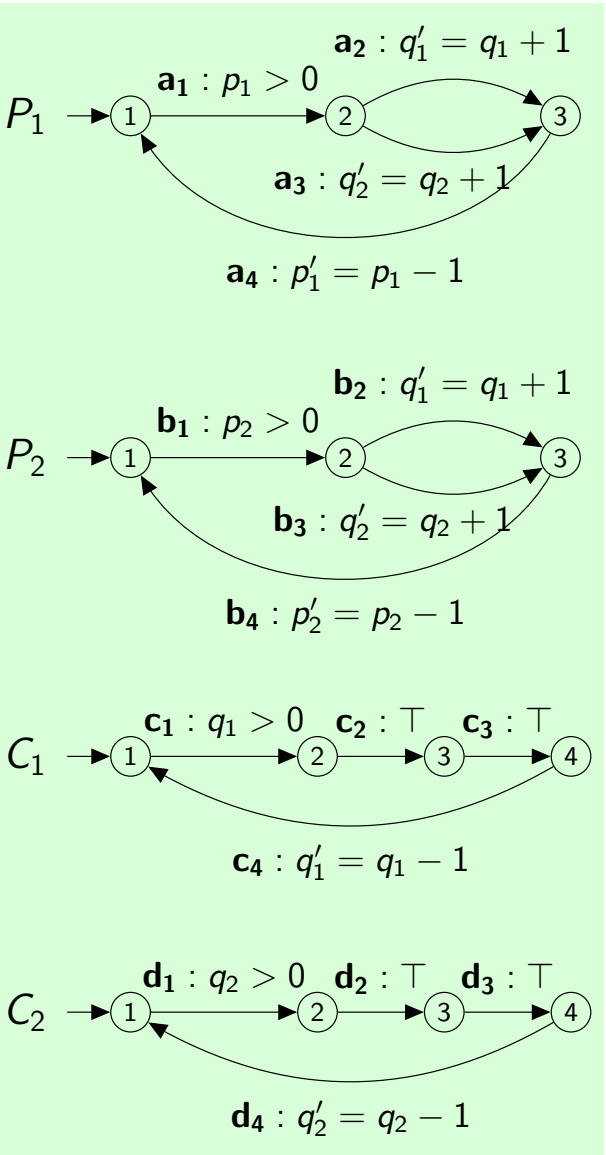


Producer-Consumer: a causal proof

$(\text{ some thread })^{\omega}$



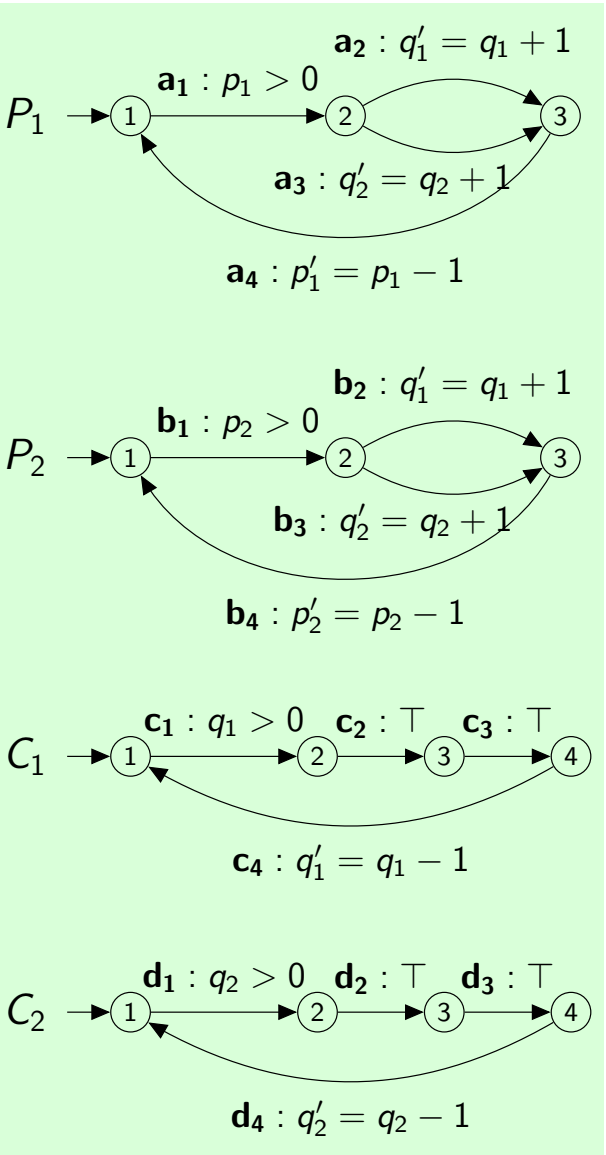
Producer-Consumer: a causal proof



(some thread)$^{\omega}$

Who runs forever?

Producer-Consumer: a causal proof

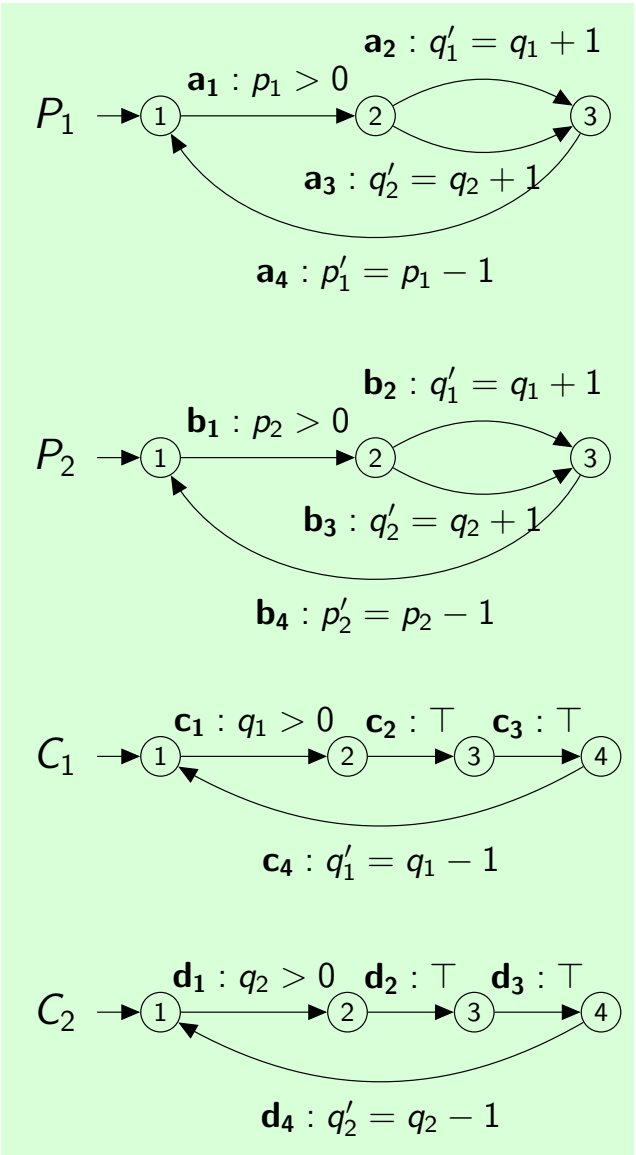


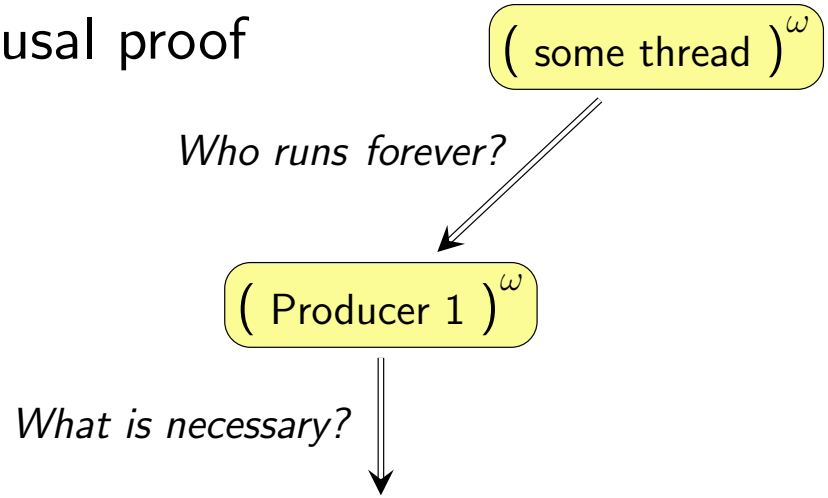
(some thread)$^\omega$

Who runs forever?

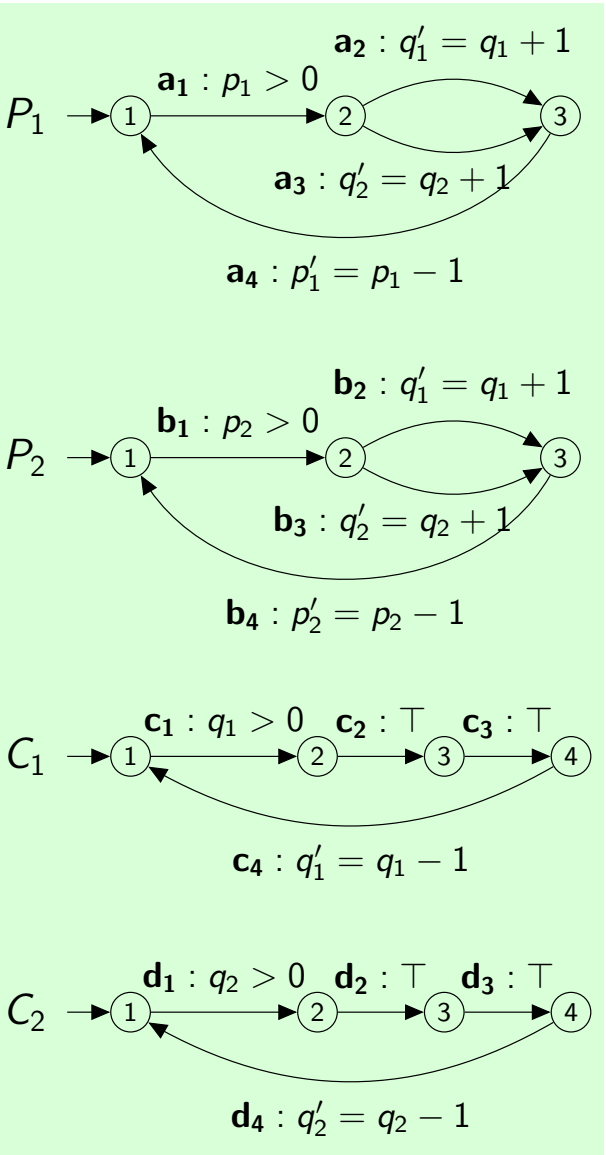
(Producer 1)$^\omega$

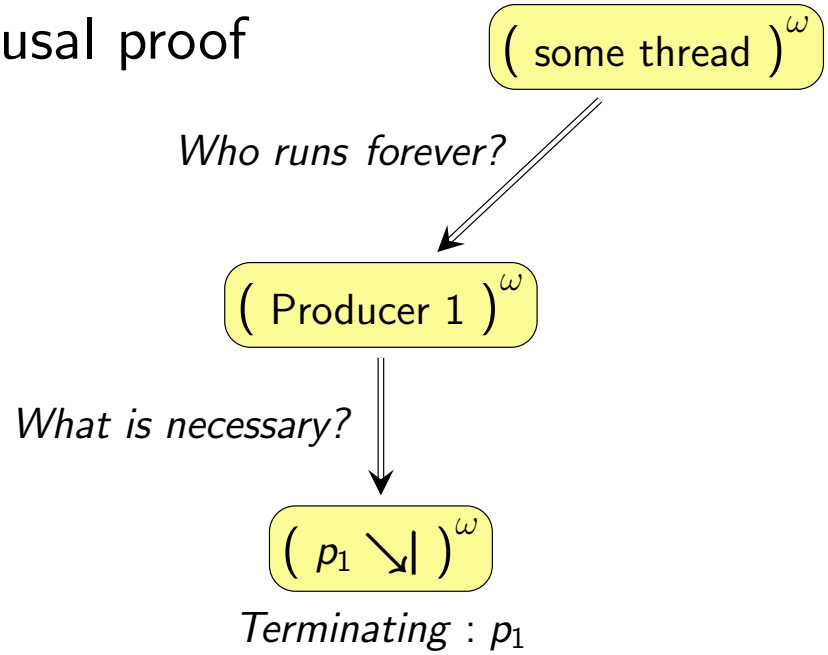
Producer-Consumer: a causal proof



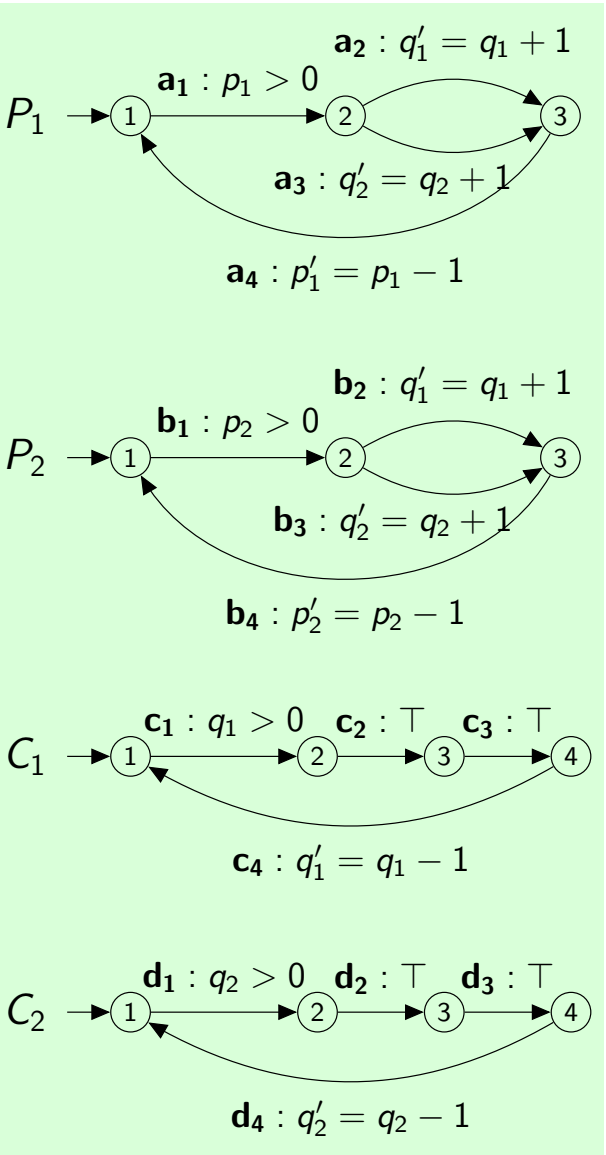


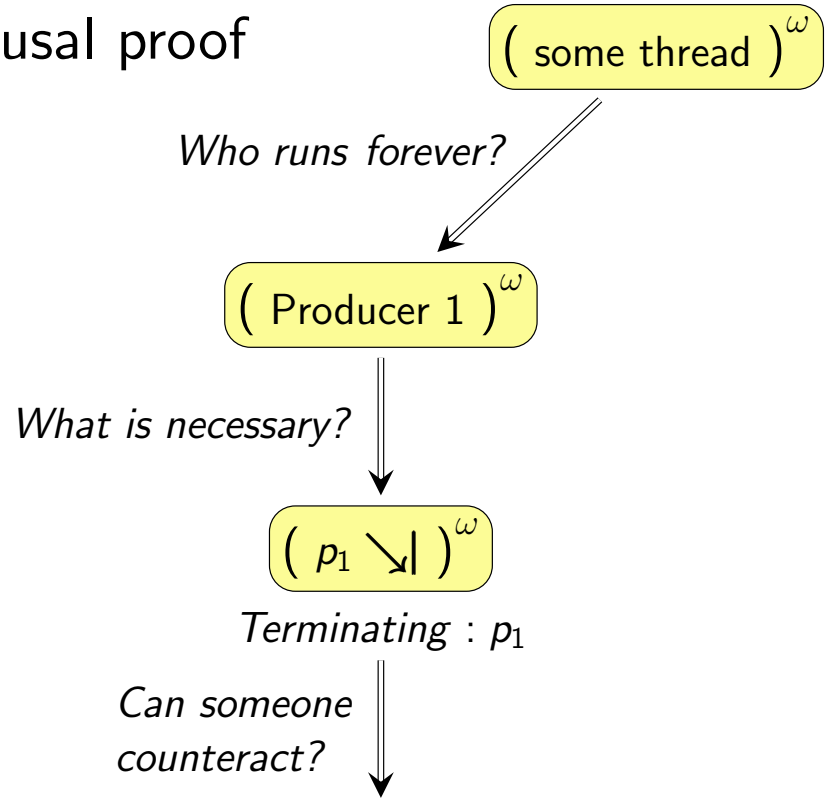
Producer-Consumer: a causal proof



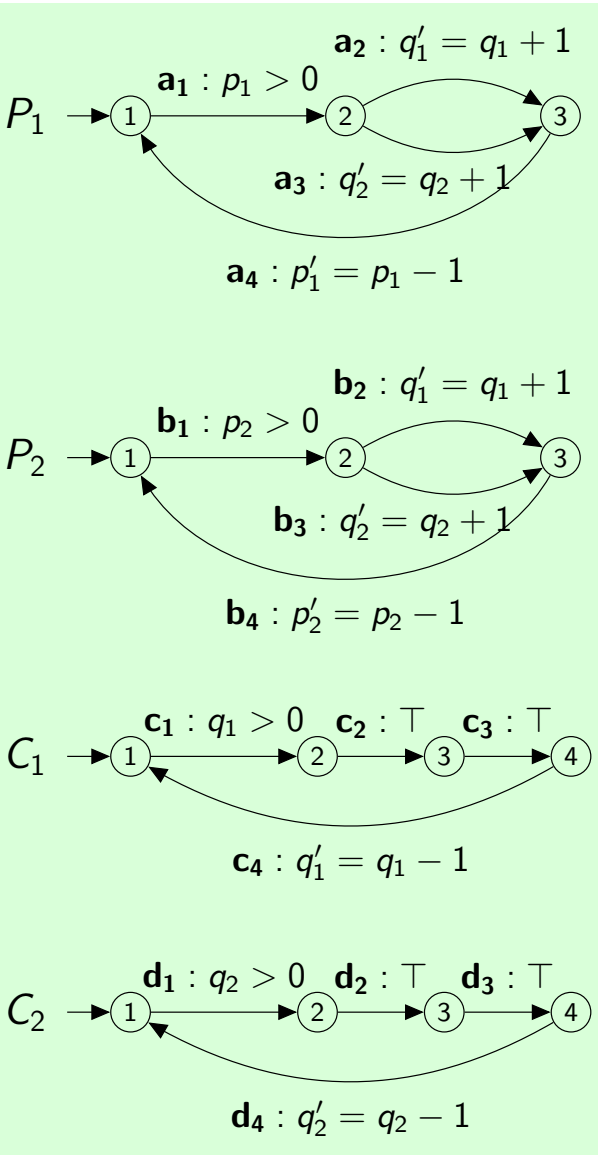


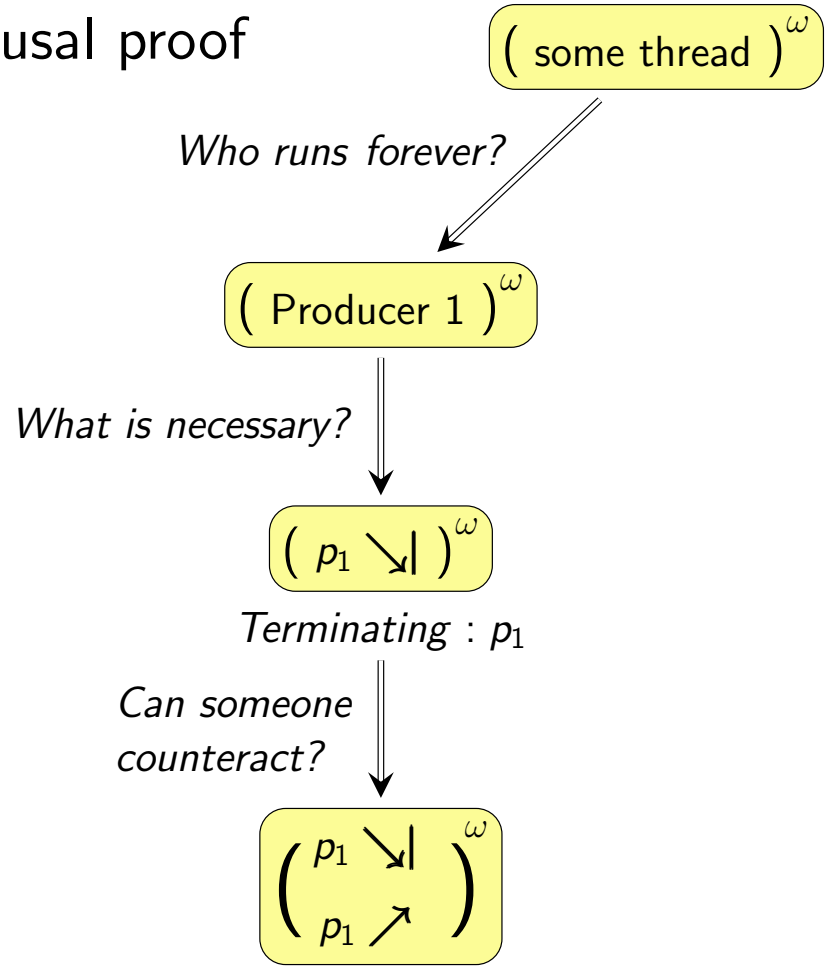
Producer-Consumer: a causal proof



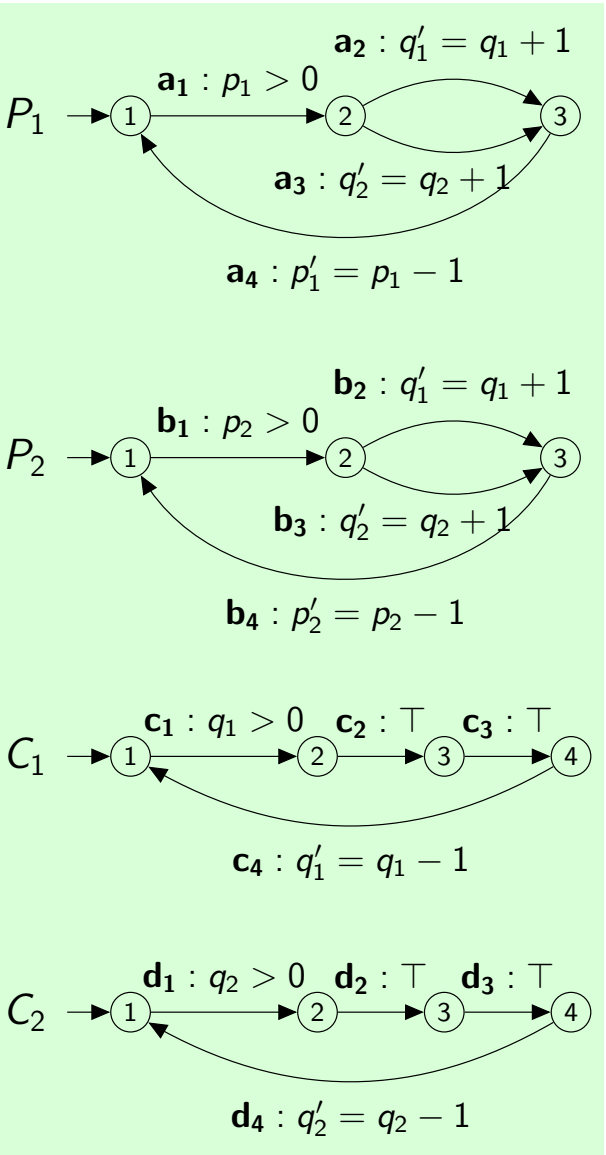


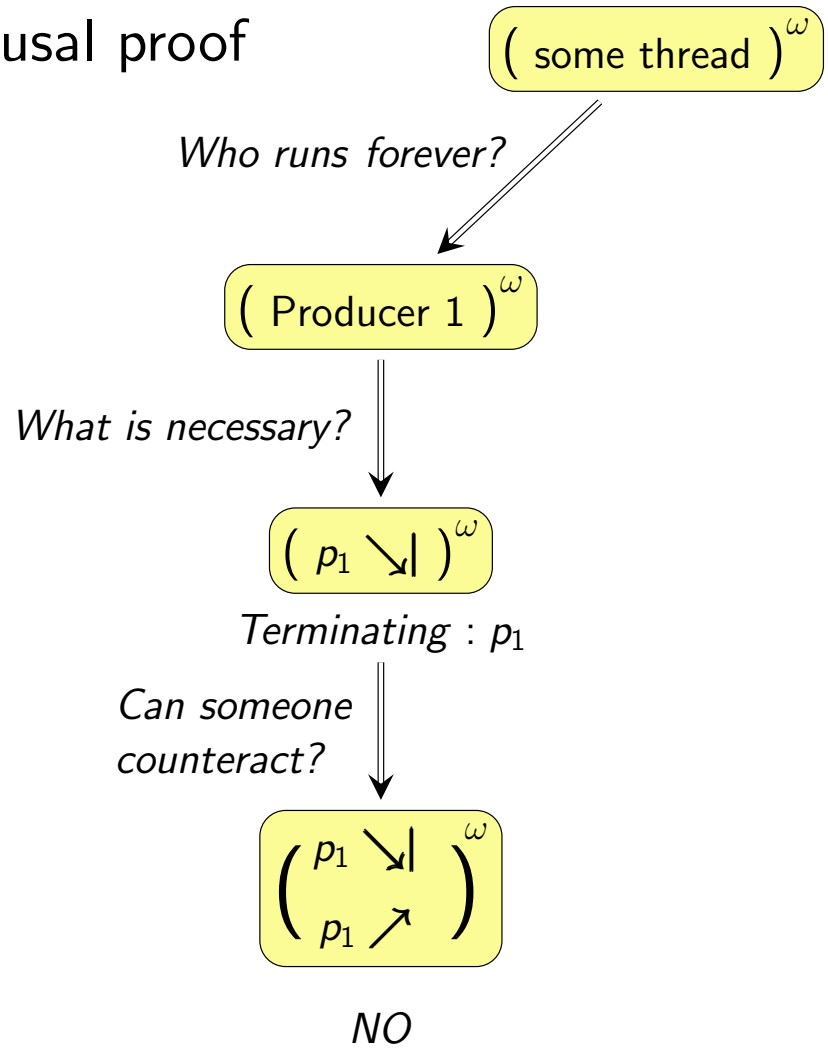
Producer-Consumer: a causal proof



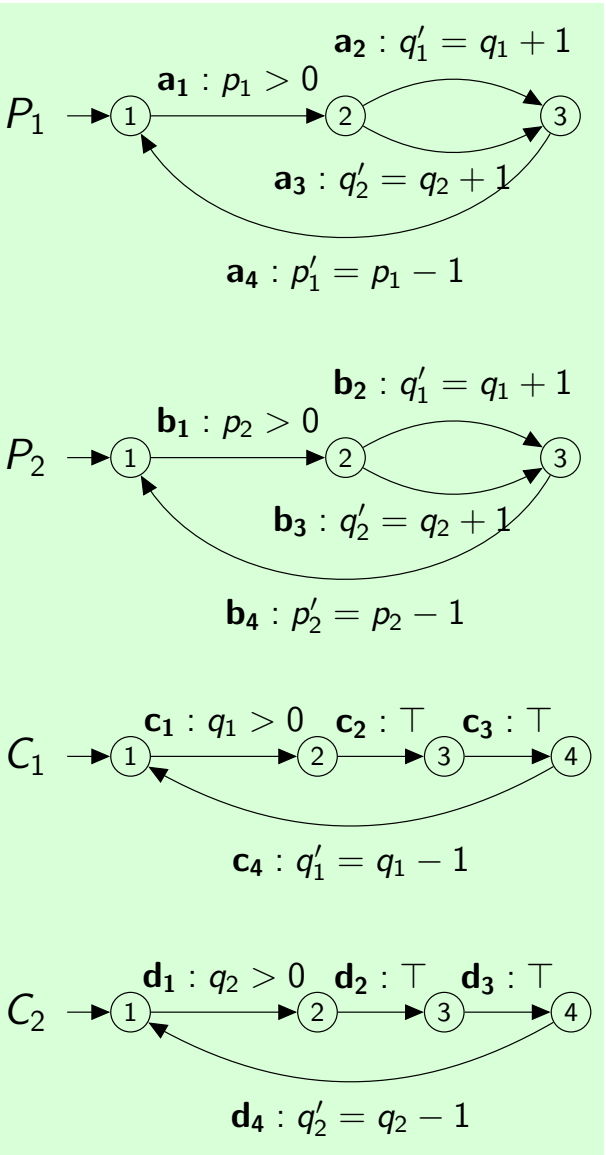


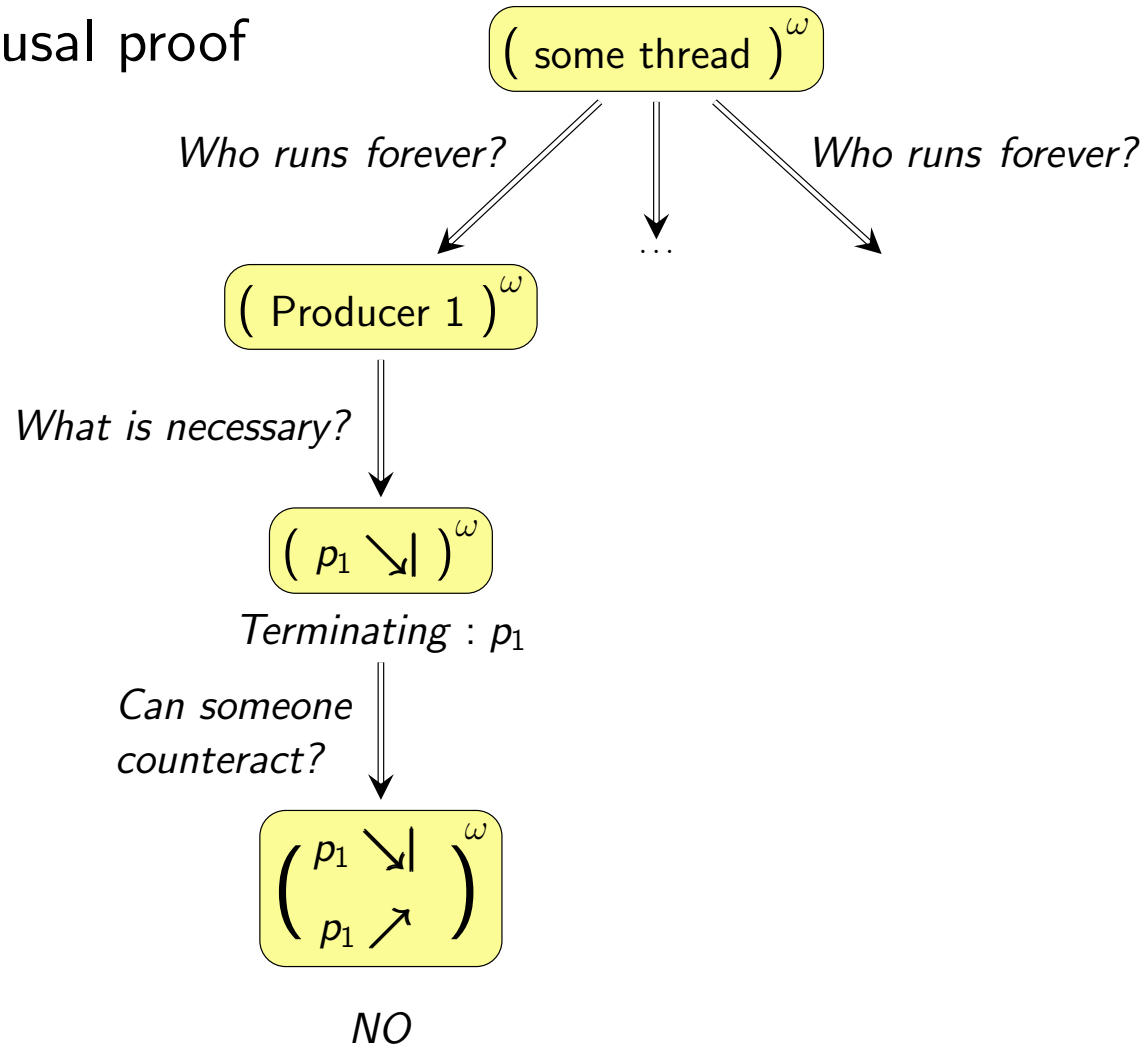
Producer-Consumer: a causal proof



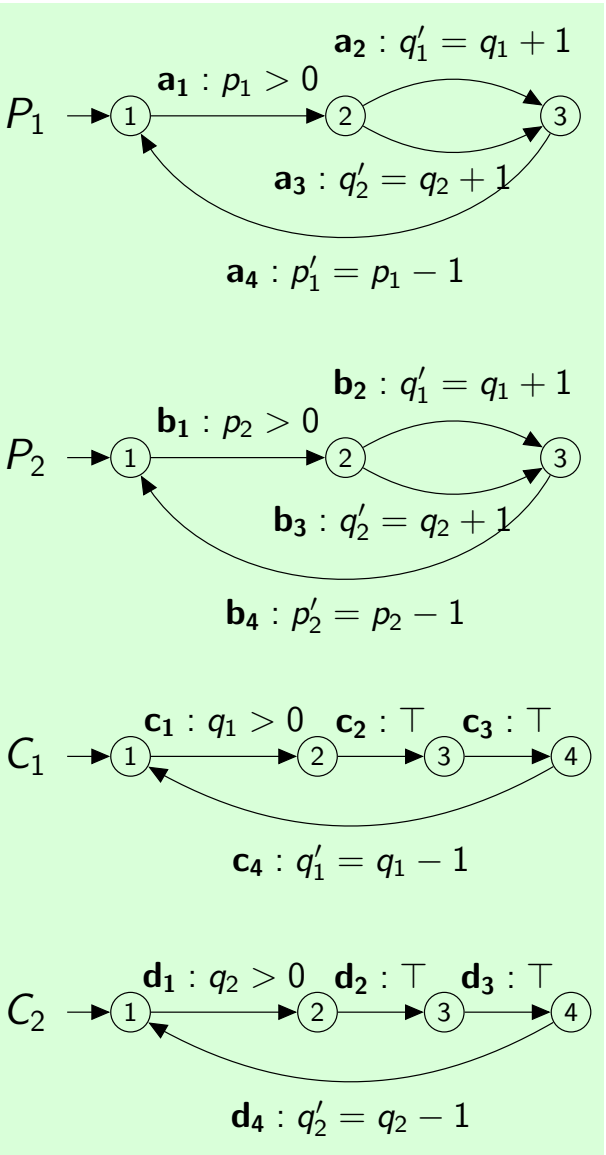


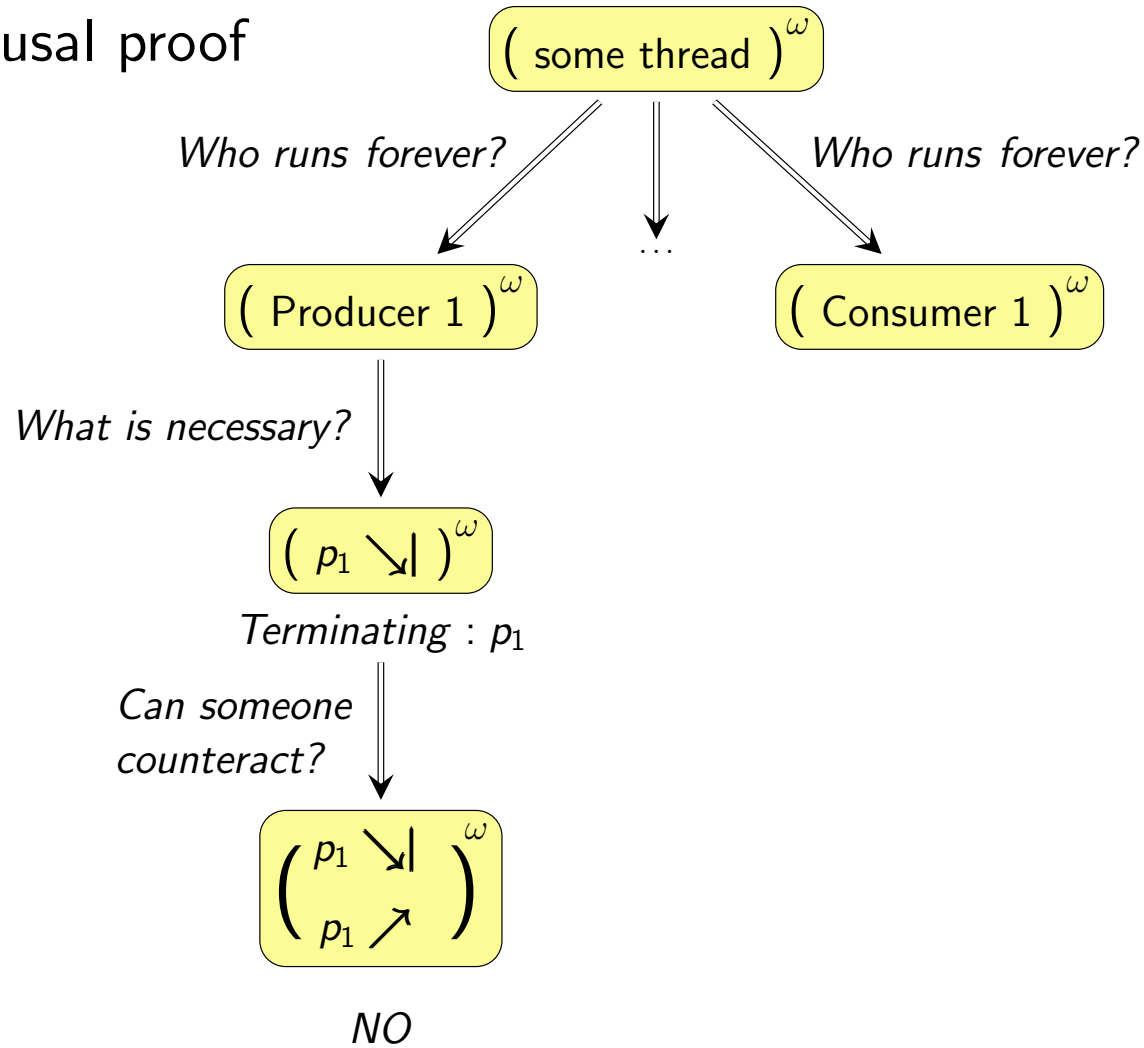
Producer-Consumer: a causal proof



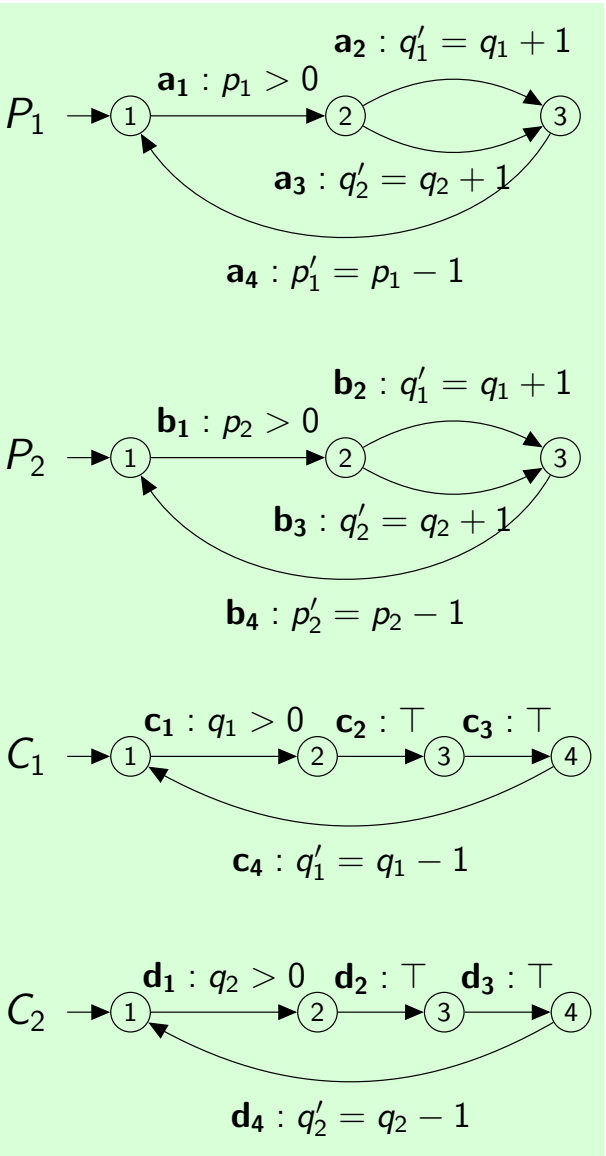


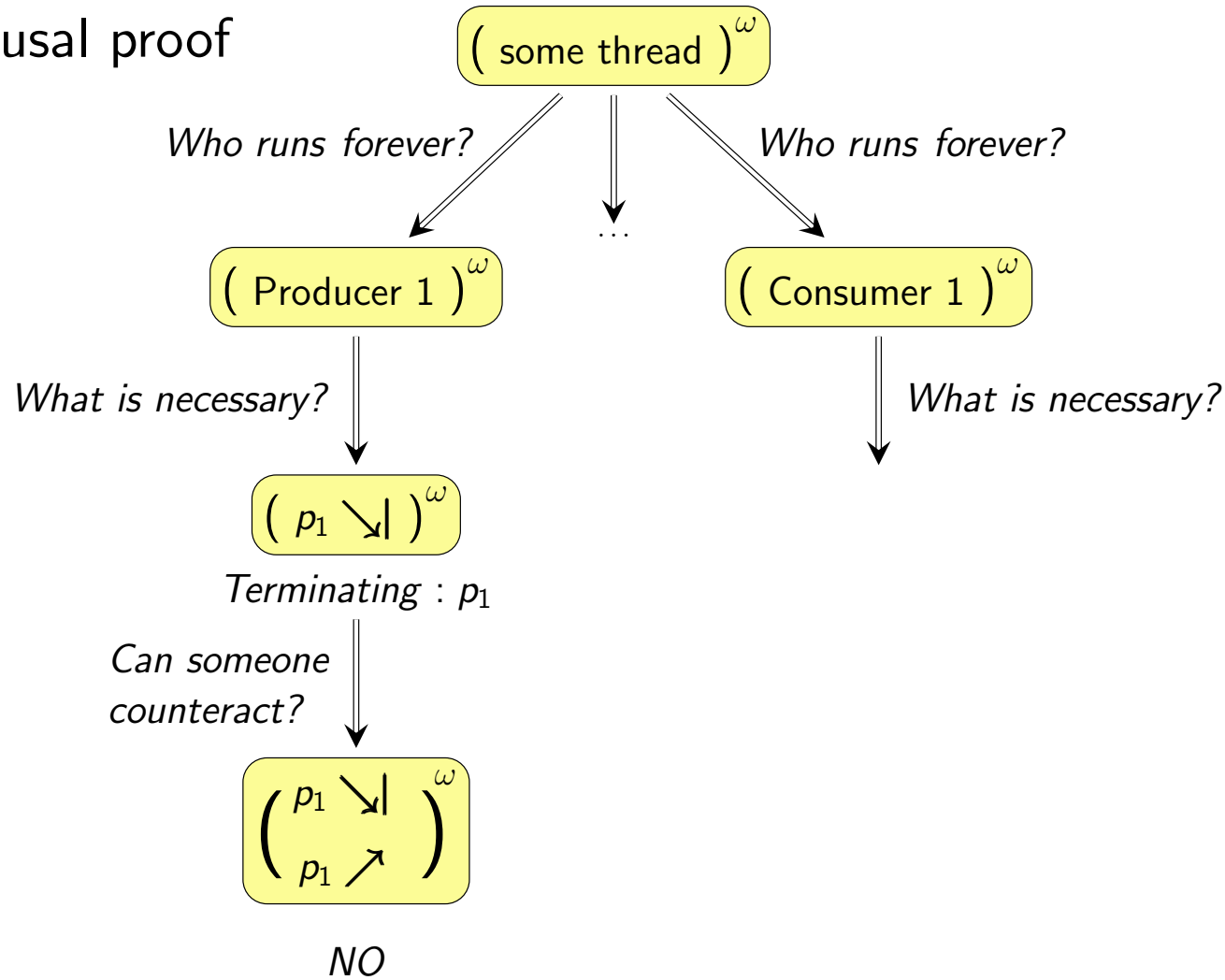
Producer-Consumer: a causal proof



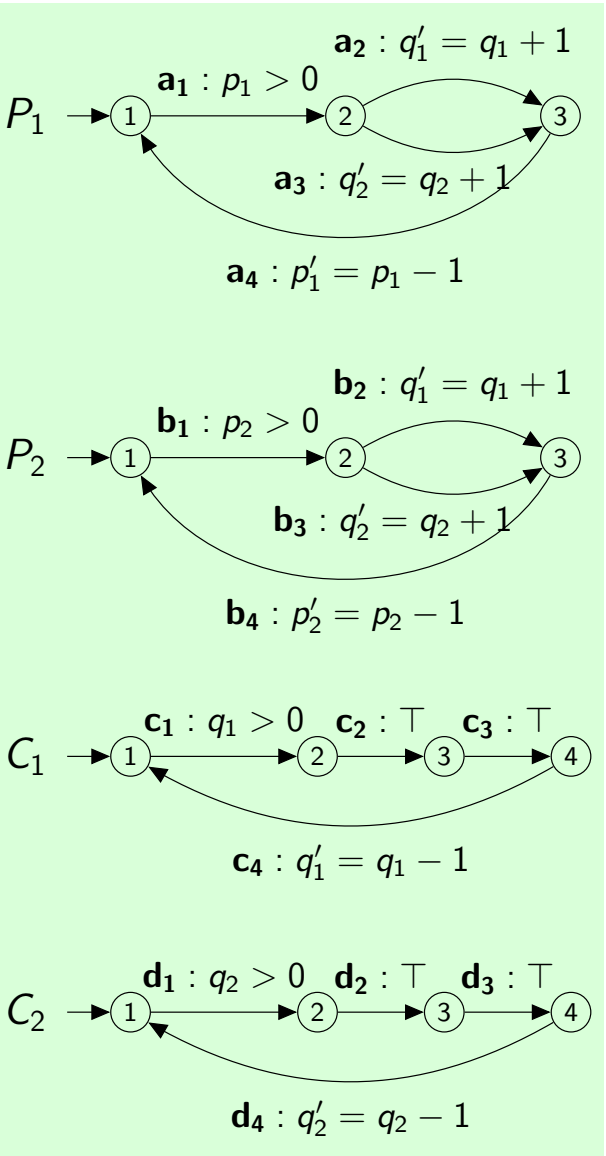


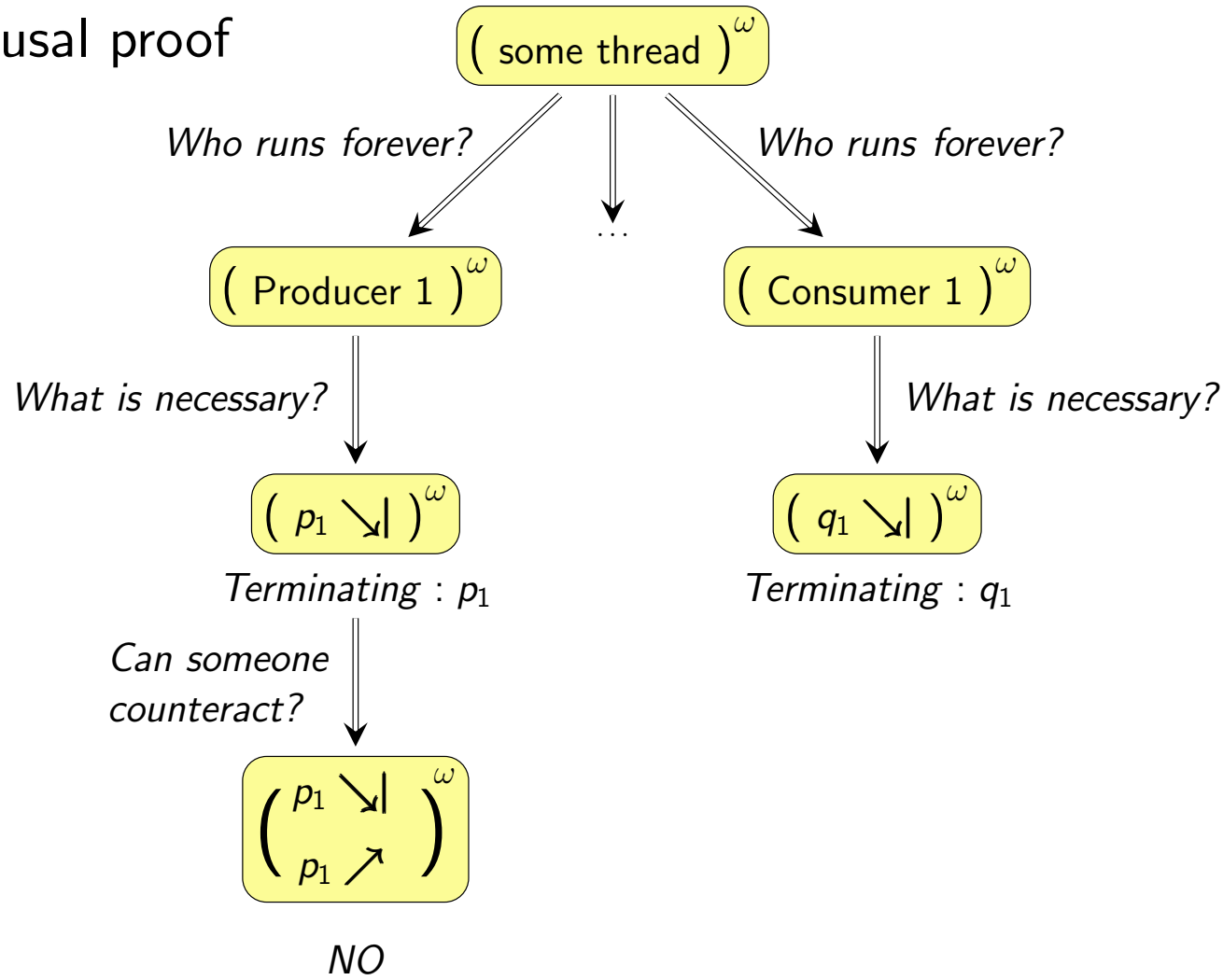
Producer-Consumer: a causal proof



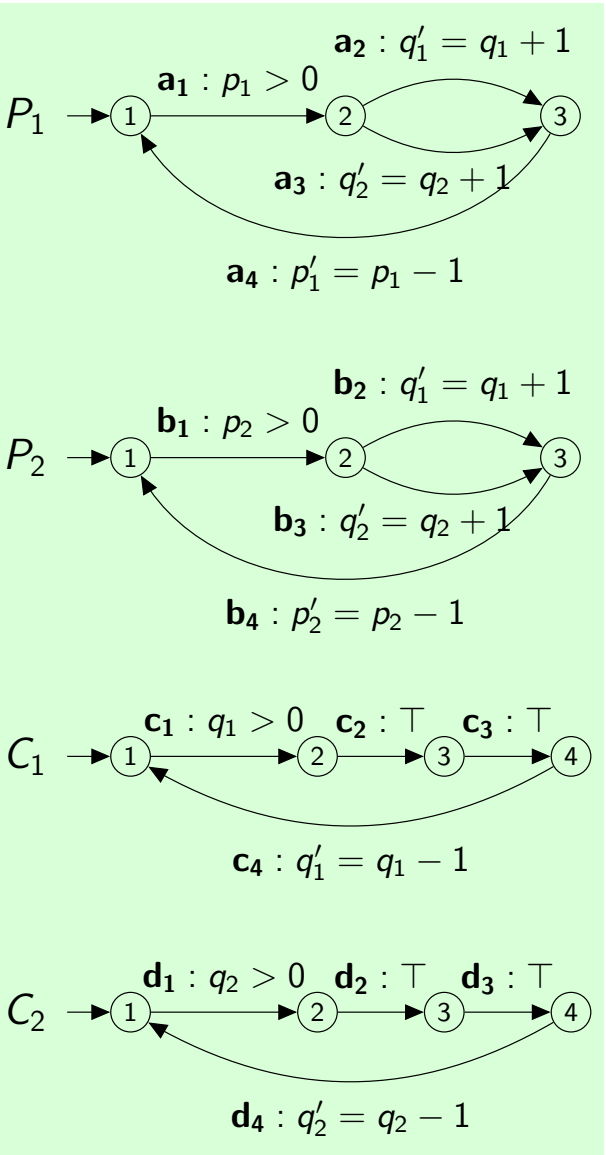


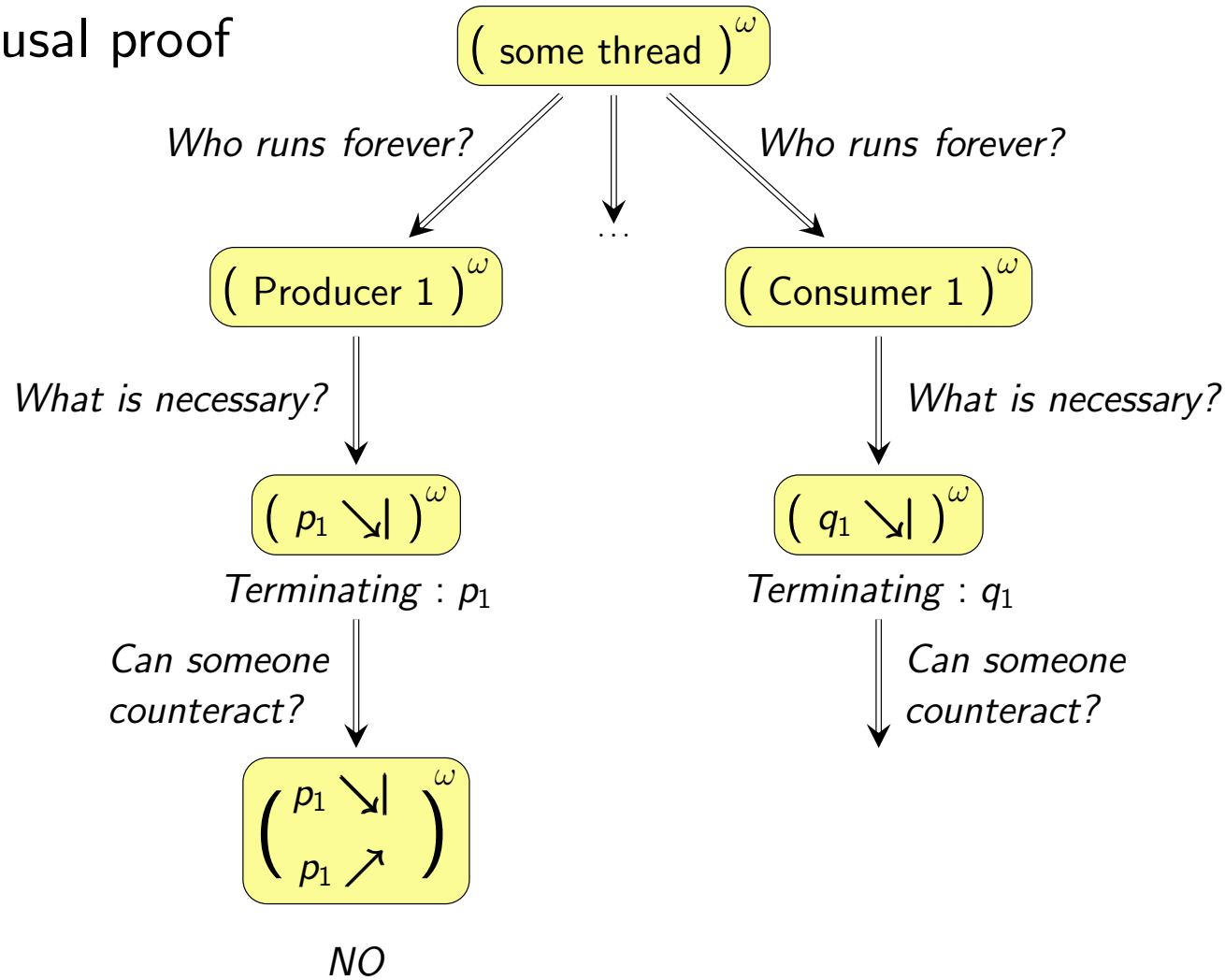
Producer-Consumer: a causal proof



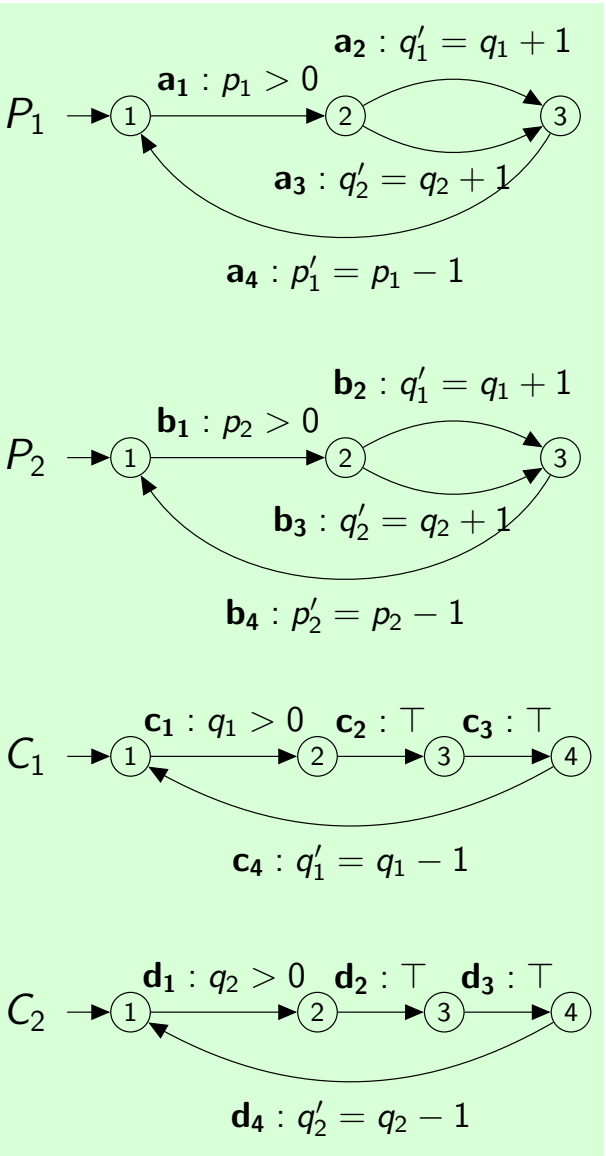


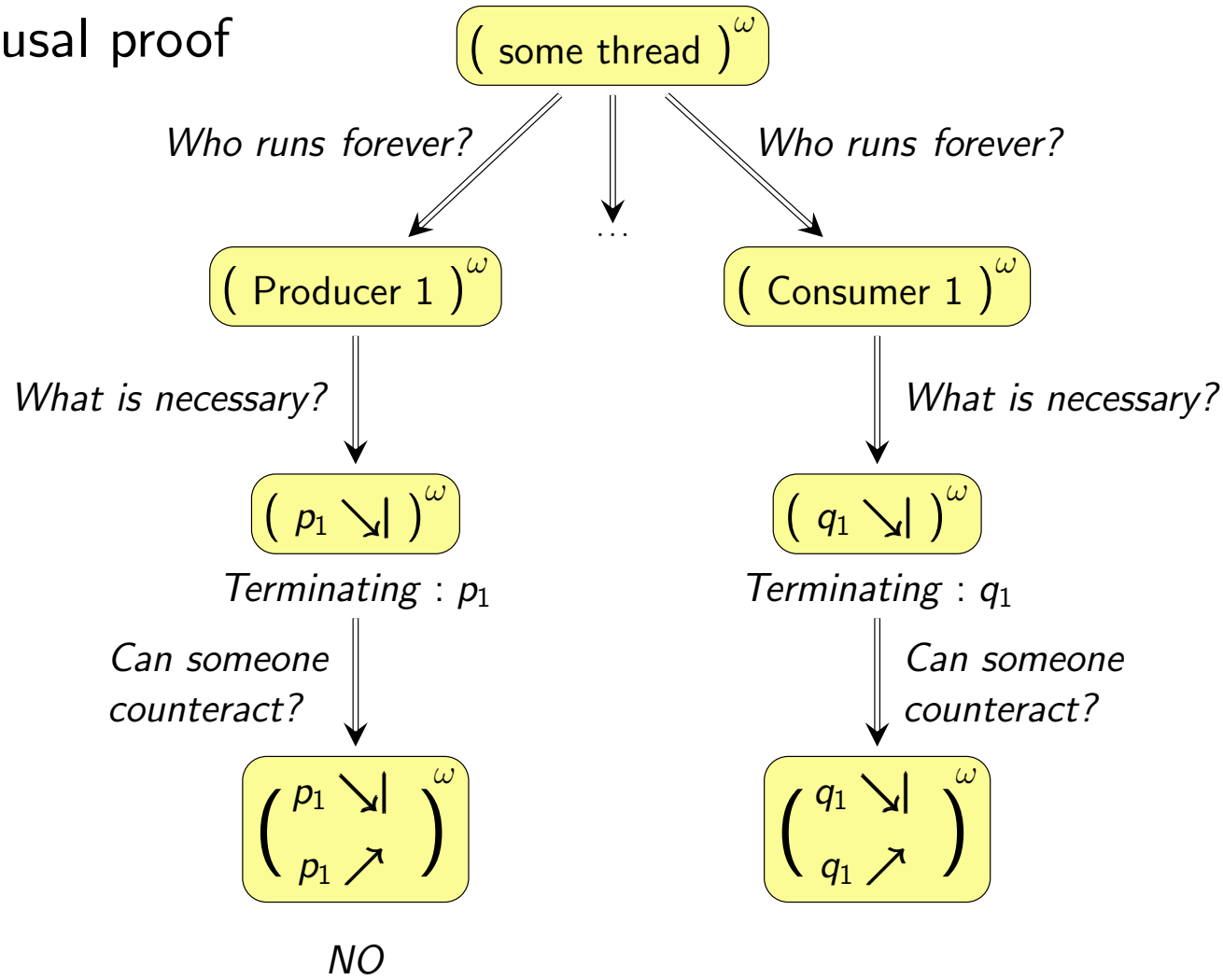
Producer-Consumer: a causal proof



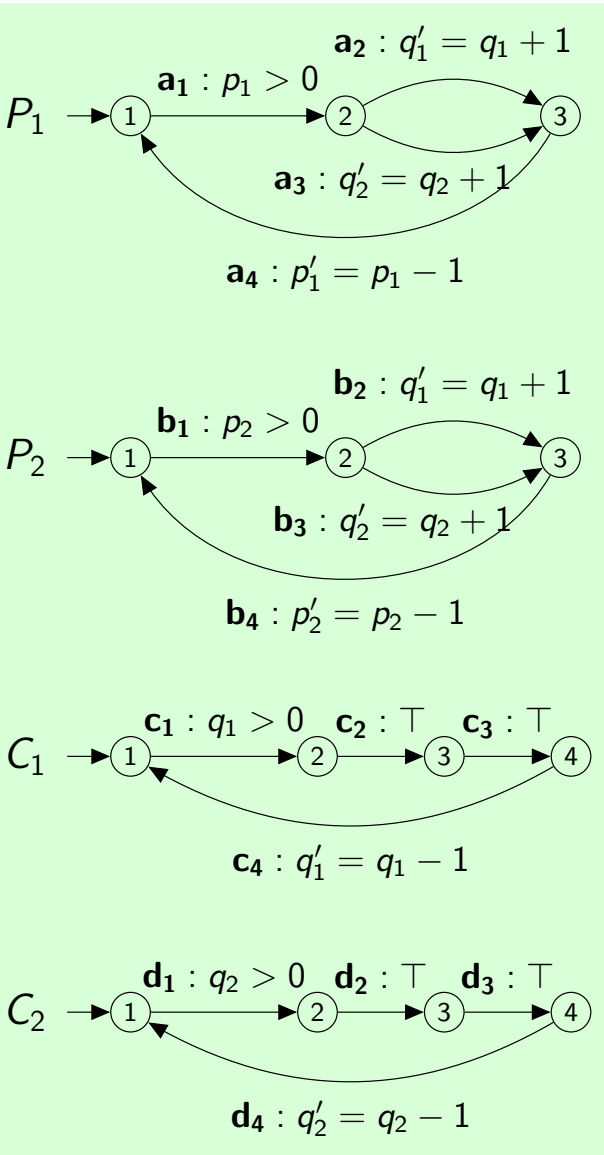


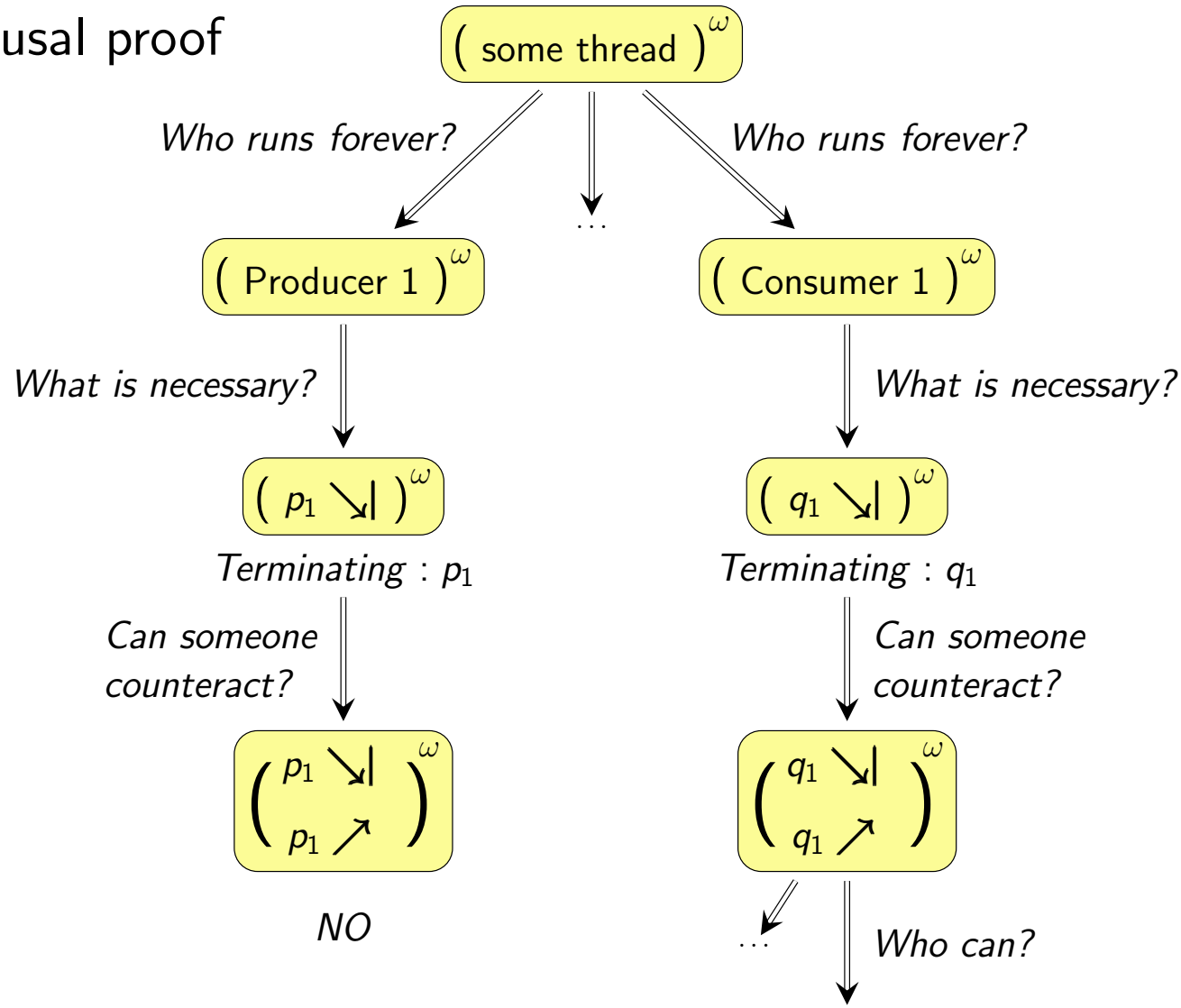
Producer-Consumer: a causal proof



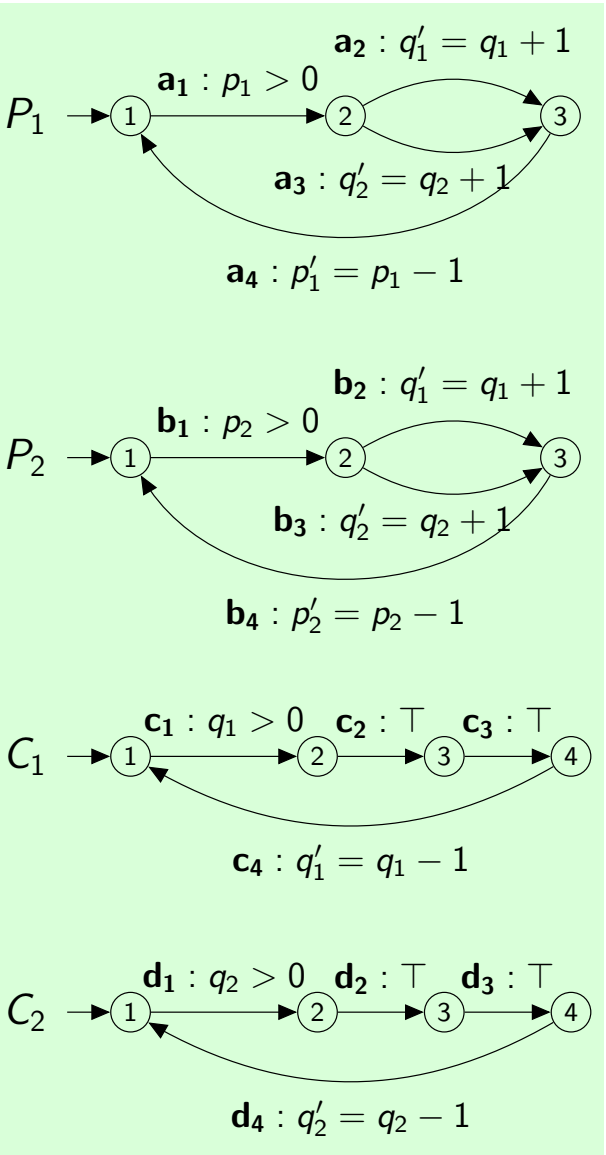


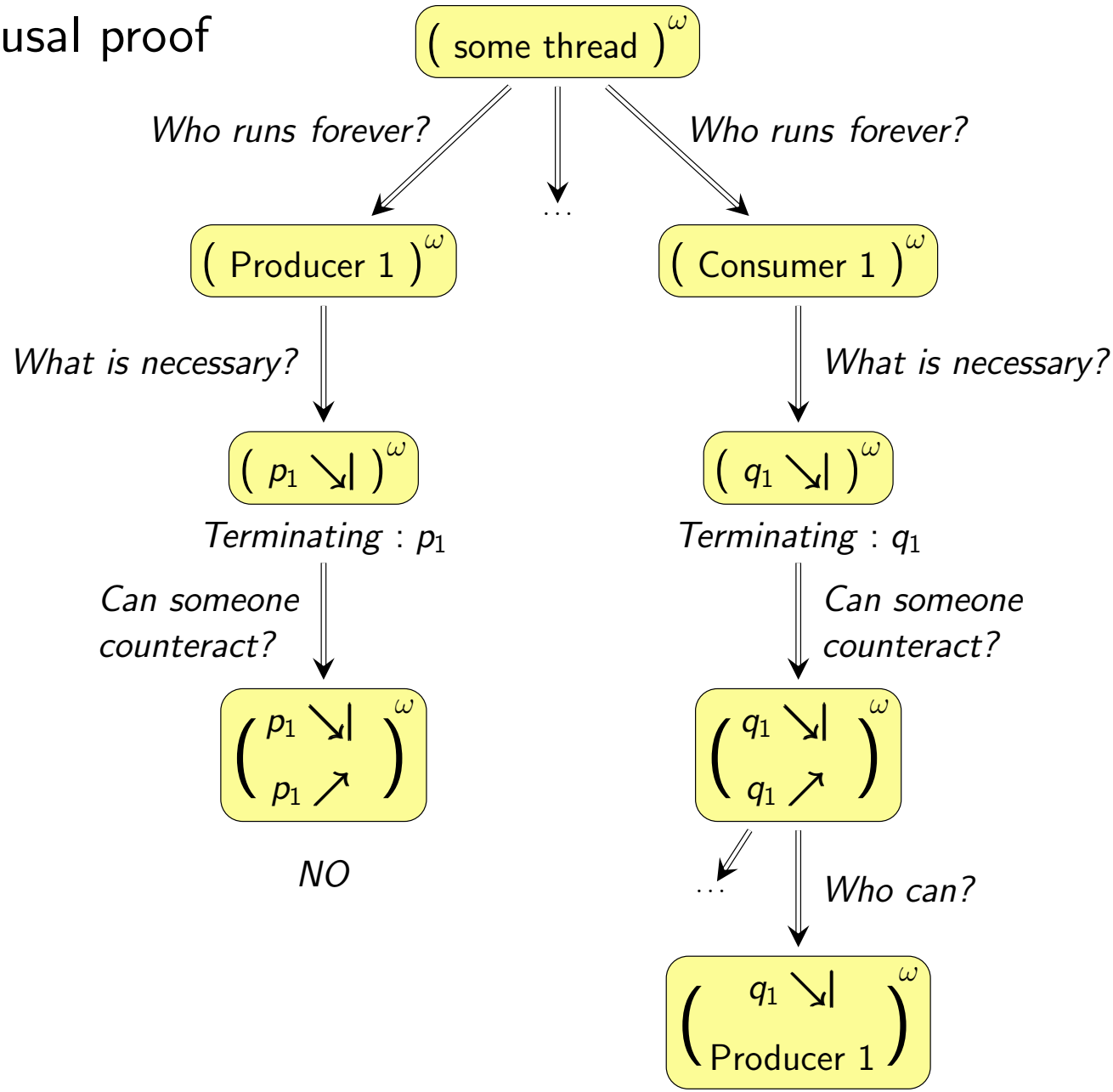
Producer-Consumer: a causal proof



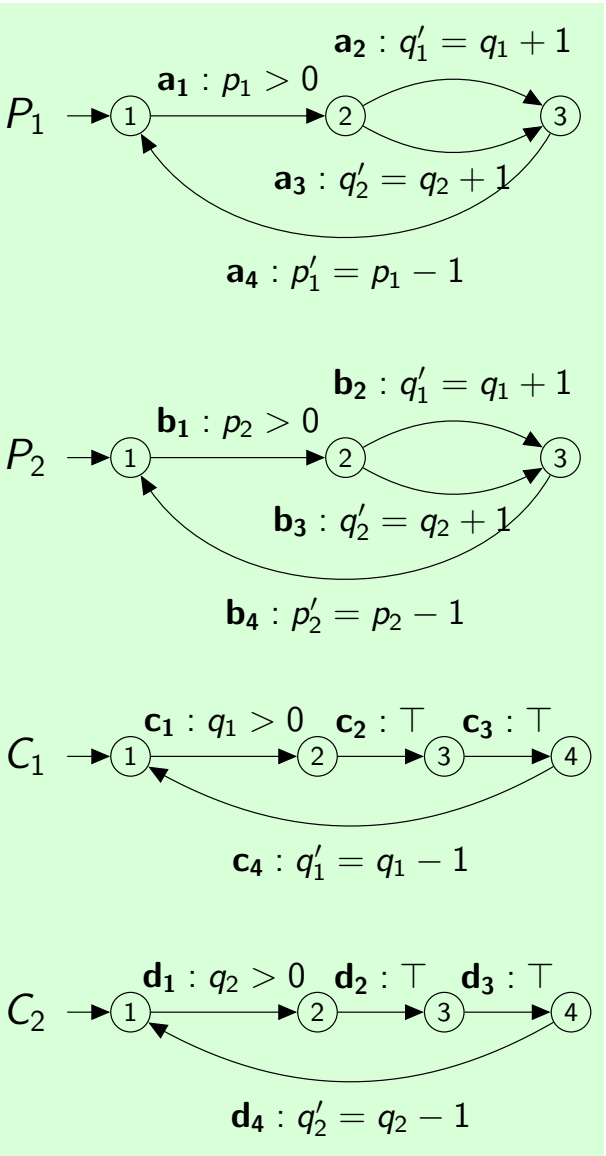


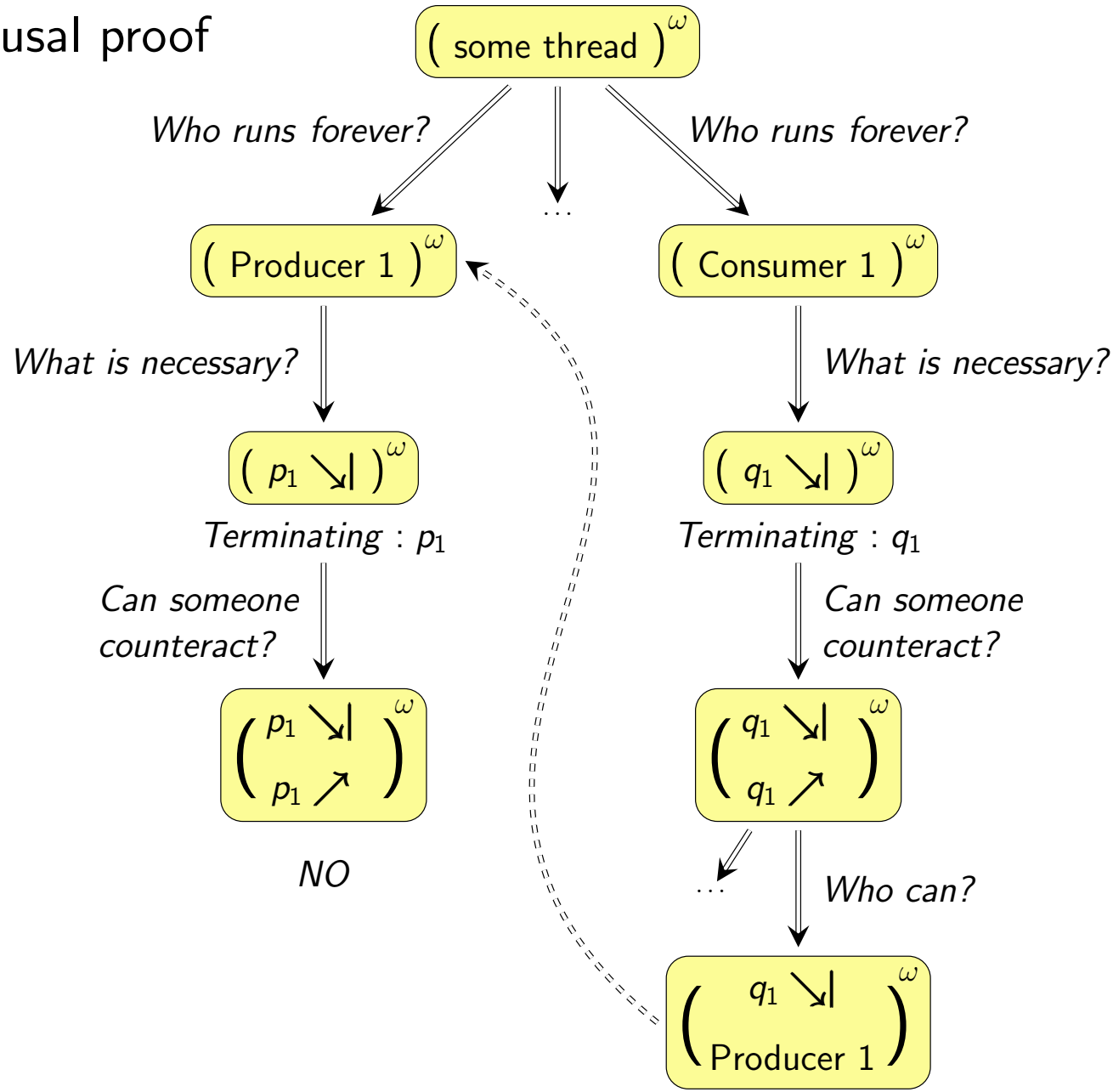
Producer-Consumer: a causal proof





Producer-Consumer: a causal proof





Finite concurrent traces

Transition system $\mathcal{S} = \langle V, I, T \rangle$

- V: variables
- $I \in \Phi(V')$: initialization
- $T \subseteq \Phi(V \cup V')$: transitions

Finite concurrent traces

- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$

Finite concurrent traces

- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Finite concurrent traces

Finite trace $\mathcal{A} = \langle N, E, \nu, \eta \rangle$

- $\langle N, E \rangle$ is a DAG
- $\nu : N \to \Phi(V \cup V')$
- $\eta : E \to \Phi(V \cup V')$

- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Finite concurrent traces

Finite trace $\mathcal{A} = \langle N, E, \nu, \eta \rangle$

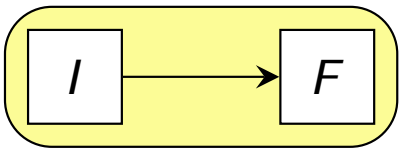
- $\langle N, E \rangle$ is a DAG
- $\nu : N \to \Phi(V \cup V')$
- $\eta : E \to \Phi(V \cup V')$

- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Language of a finite concurrent trace

A set of system runs such that a *linearization* of a concurrent trace can be mapped into a *subsequence* of a run, *respecting constraints*

Finite concurrent traces

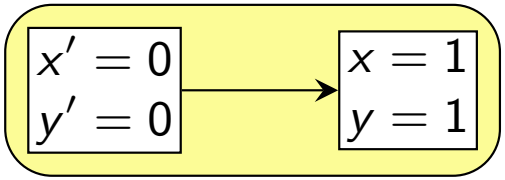


- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Language of a finite concurrent trace

A set of system runs such that a *linearization* of a concurrent trace can be mapped into a *subsequence* of a run, *respecting constraints*

Finite concurrent traces



- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

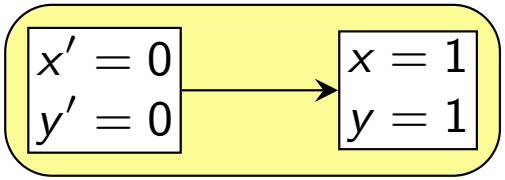
Accepted runs

- $I, \boldsymbol{x^+}, \boldsymbol{y^+}, F$
- $I, \boldsymbol{y^+}, \boldsymbol{x^+}, F$
- $I, \boldsymbol{y^+}, \boldsymbol{x^+}, \boldsymbol{x^-}, \boldsymbol{x^+}, F$
- $\ldots$

Rejected runs

- $I, \boldsymbol{x^+}, F$
- $I, \boldsymbol{x^+}, \boldsymbol{y^+}, \boldsymbol{x^+}, F$
- $I, \boldsymbol{x^-}, \boldsymbol{y^-}, F$
- $\ldots$

Finite concurrent traces



- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

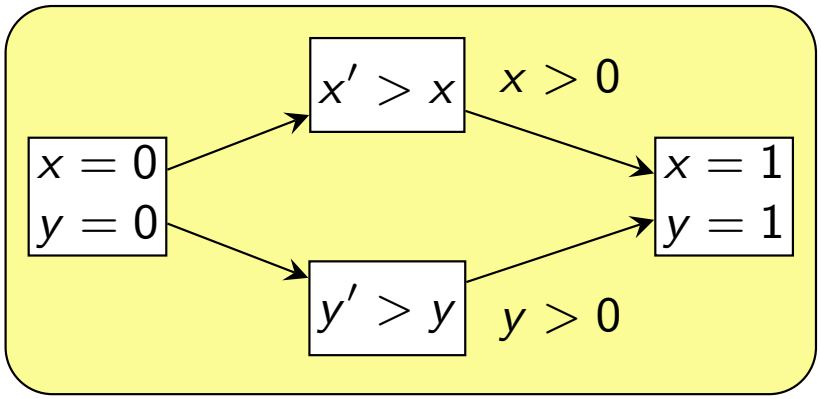
Accepted runs

- $I, \boldsymbol{x^+}, \boldsymbol{y^+}, F$
- $I, \boldsymbol{y^+}, \boldsymbol{x^+}, F$
- $I, \boldsymbol{y^+}, \boldsymbol{x^+}, \boldsymbol{x^-}, \boldsymbol{x^+}, F$
- $\ldots$

Rejected runs

- $I, \boldsymbol{x^+}, F$
- $I, \boldsymbol{x^+}, \boldsymbol{y^+}, \boldsymbol{x^+}, F$
- $I, \boldsymbol{x^-}, \boldsymbol{y^-}, F$
- $\ldots$

Finite concurrent traces



- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

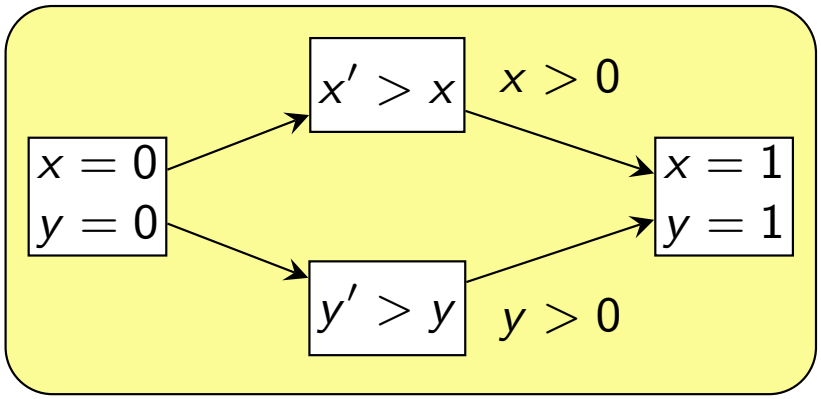
Accepted runs

- $I, \boldsymbol{x^+}, \boldsymbol{y^+}, F$
- $I, \boldsymbol{y^+}, \boldsymbol{x^+}, F$
- $I, \boldsymbol{y^+}, \boldsymbol{x^+}, \boldsymbol{x^-}, \boldsymbol{x^+}, F$
- ...

Rejected runs

- $I, \boldsymbol{x^+}, F$
- $I, \boldsymbol{x^+}, \boldsymbol{y^+}, \boldsymbol{x^+}, F$
- $I, \boldsymbol{x^-}, \boldsymbol{y^-}, F$
- ...

Finite concurrent traces

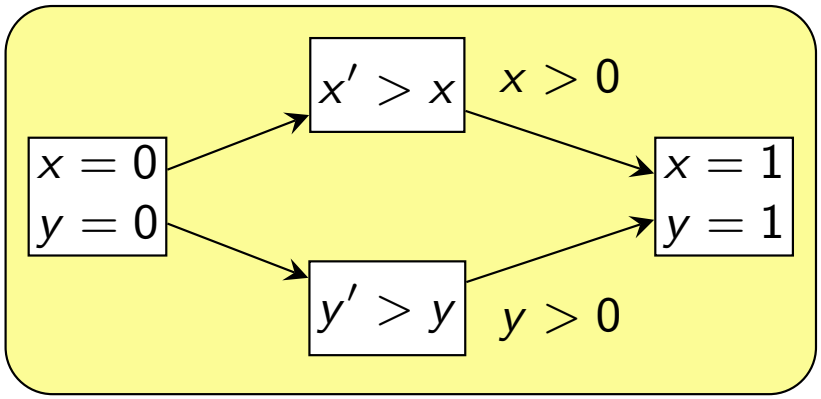


- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Accepted run

I $\quad \boldsymbol{x^+} \quad$ $\boldsymbol{y^+}$ $\quad F$

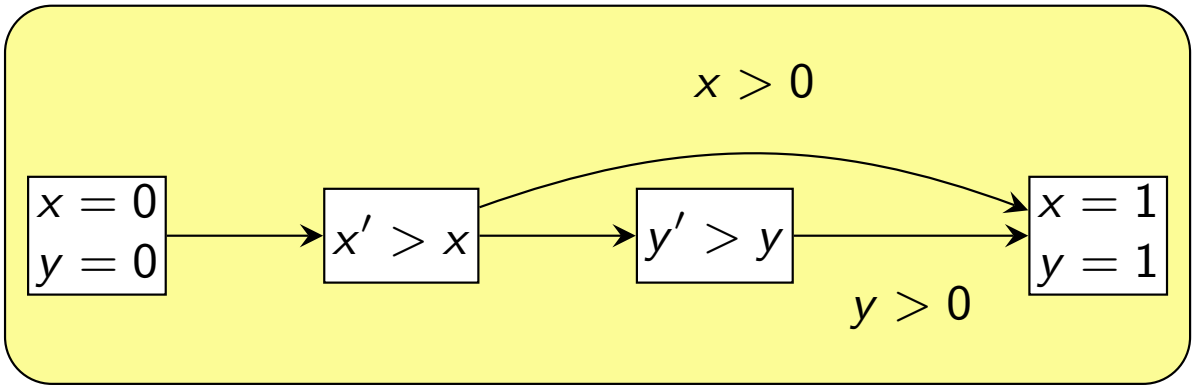
Finite concurrent traces



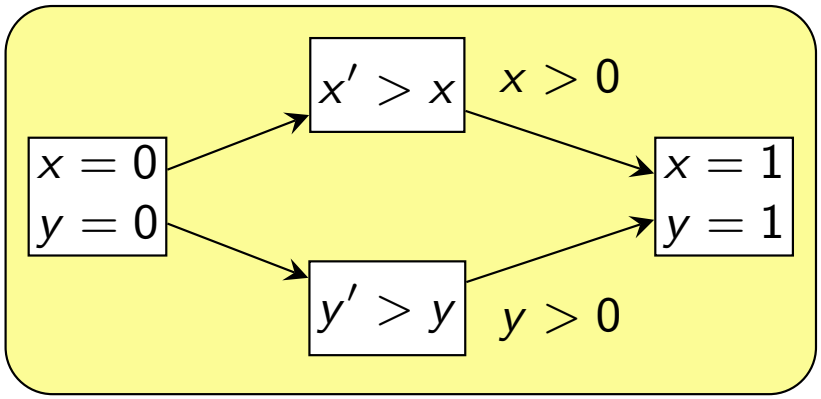
- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon\ x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon\ x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon\ y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon\ y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Accepted run

I $\quad$ $\boldsymbol{x^+}$ $\quad$ $\boldsymbol{y^+}$ $\quad$ F



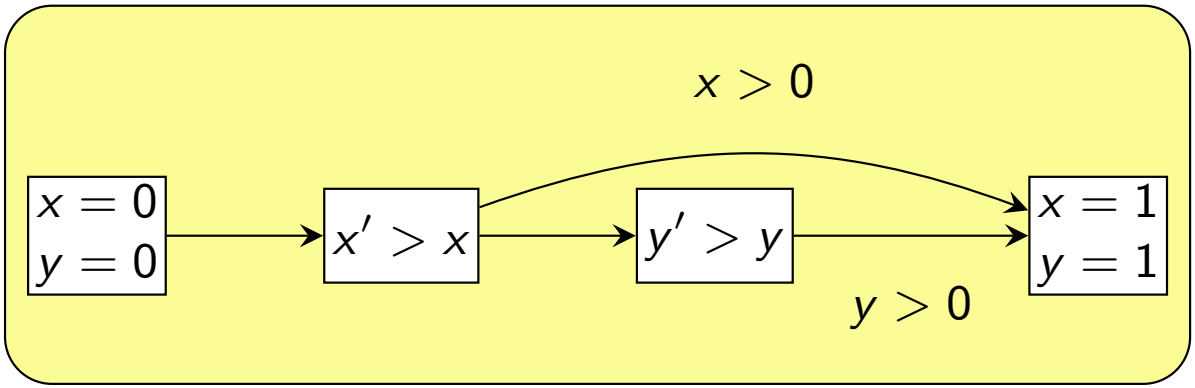
Finite concurrent traces



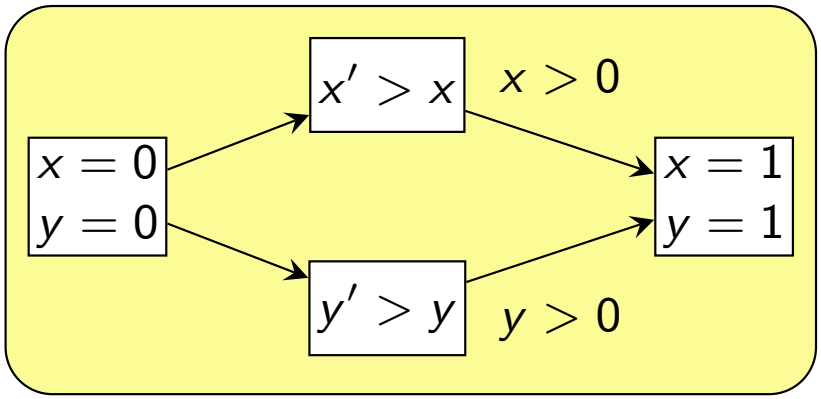
- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Accepted run

I $\quad$ $\boldsymbol{y^+}$ $\quad$ $\boldsymbol{x^+}$ $\quad$ F



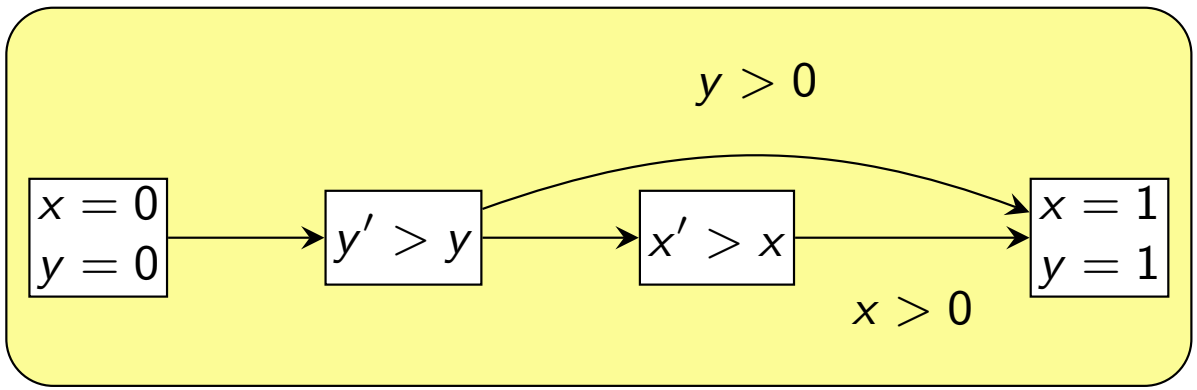
Finite concurrent traces



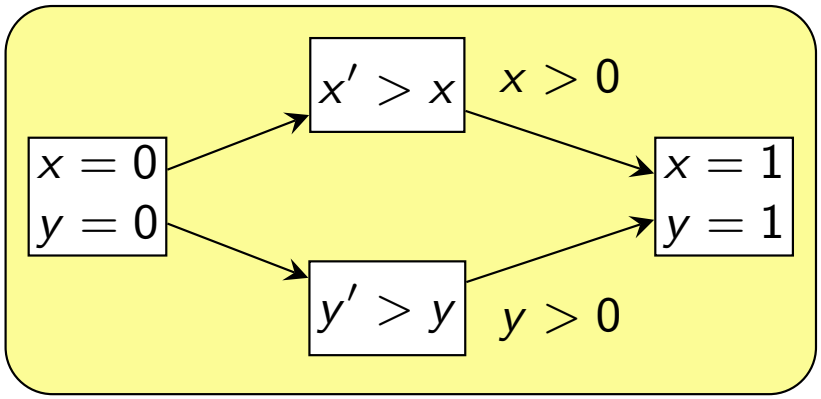
- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Accepted run

I $\quad$ $\boldsymbol{y^+}$ $\quad$ $\boldsymbol{x^+}$ $\quad$ F



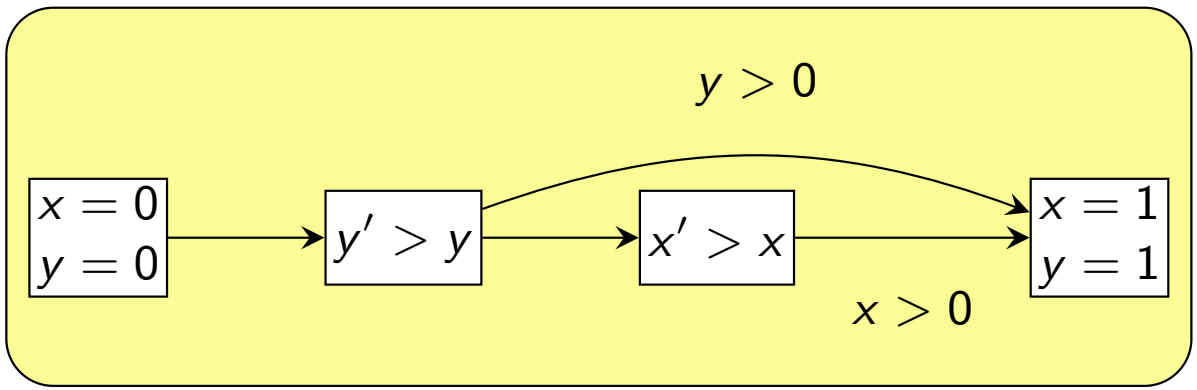
Finite concurrent traces



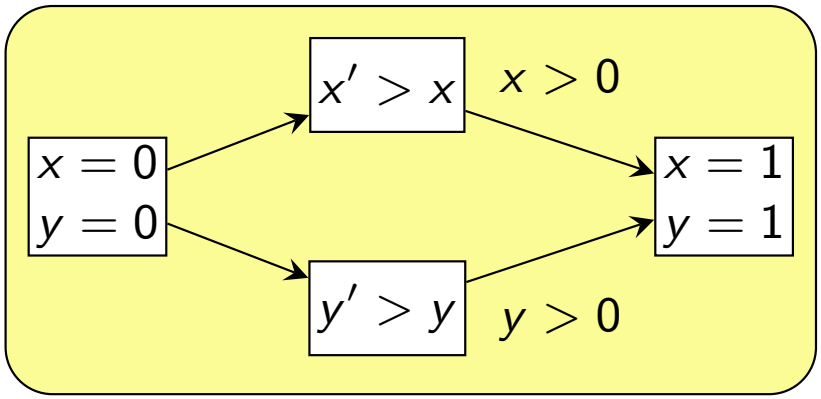
- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

Accepted run

I $\quad$ $\boldsymbol{y^+}$ $\quad$ $\boldsymbol{x^+}$ $\quad$ $\boldsymbol{x^-}$ $\quad$ $\boldsymbol{x^+}$ $\quad$ F



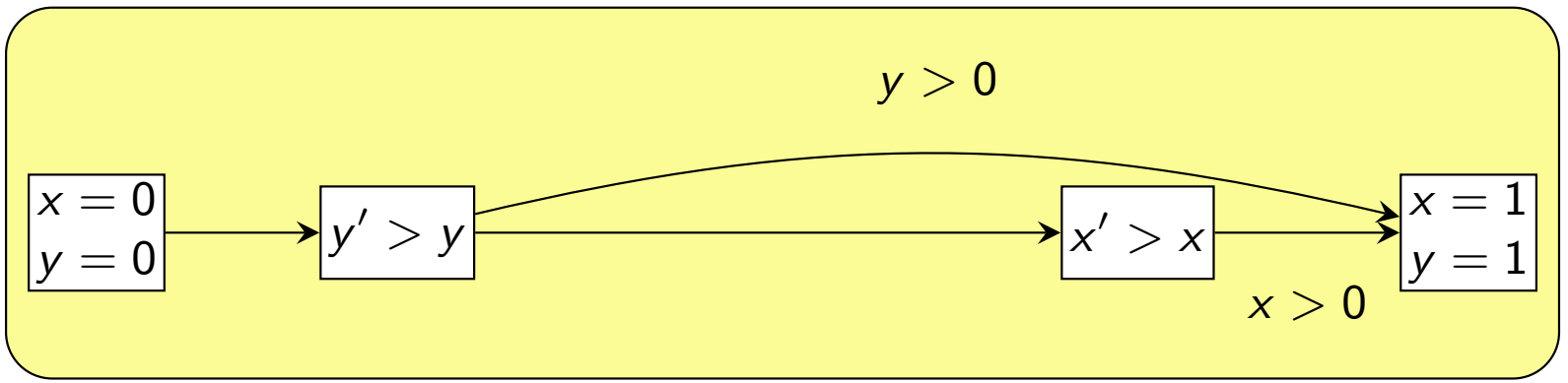
Finite concurrent traces



- $I \equiv x' = 0 \wedge y' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon x' = x + 1 \wedge y' = y$
  $\boldsymbol{x^-}\colon x' = x - 1 \wedge y' = y$
  $\boldsymbol{y^+}\colon y' = y + 1 \wedge x' = x$
  $\boldsymbol{y^-}\colon y' = y - 1 \wedge x' = x\ \}$
- $F \equiv x = 1 \wedge y = 1$

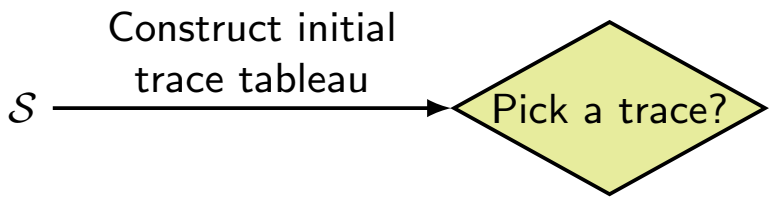
Accepted run

I $\quad \boldsymbol{y^+} \quad \boldsymbol{x^+} \quad \boldsymbol{x^-} \quad \boldsymbol{x^+} \quad F$



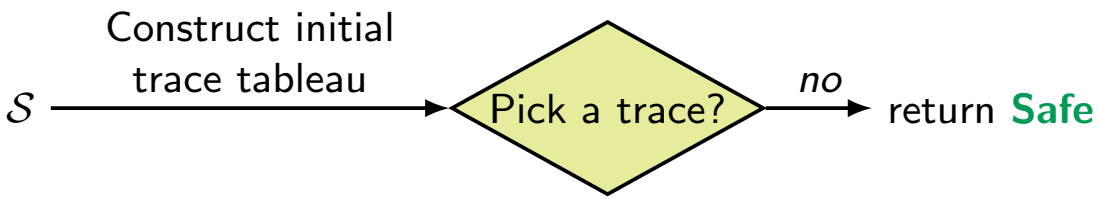
Reachability analysis algorithm

[K., Finkbeiner, CONCUR 2013]



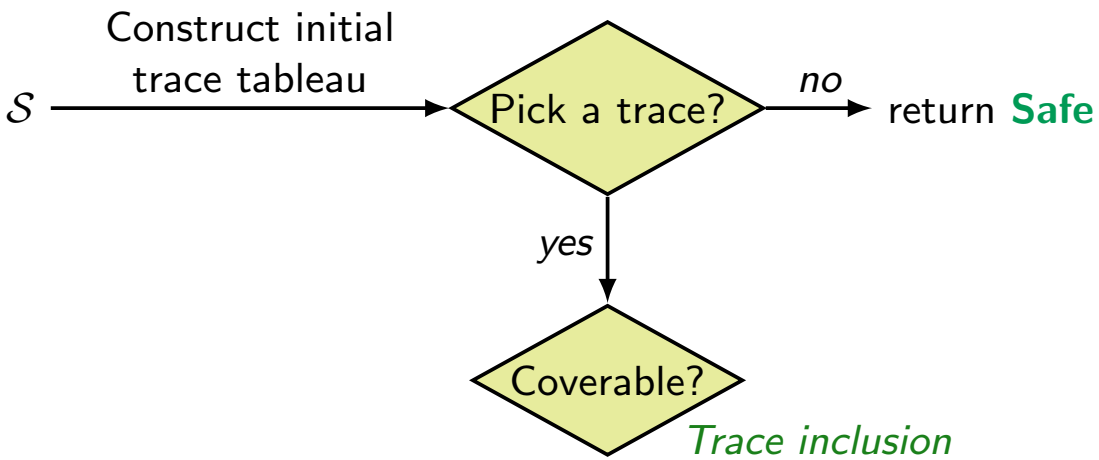
Reachability analysis algorithm

[K., Finkbeiner, CONCUR 2013]



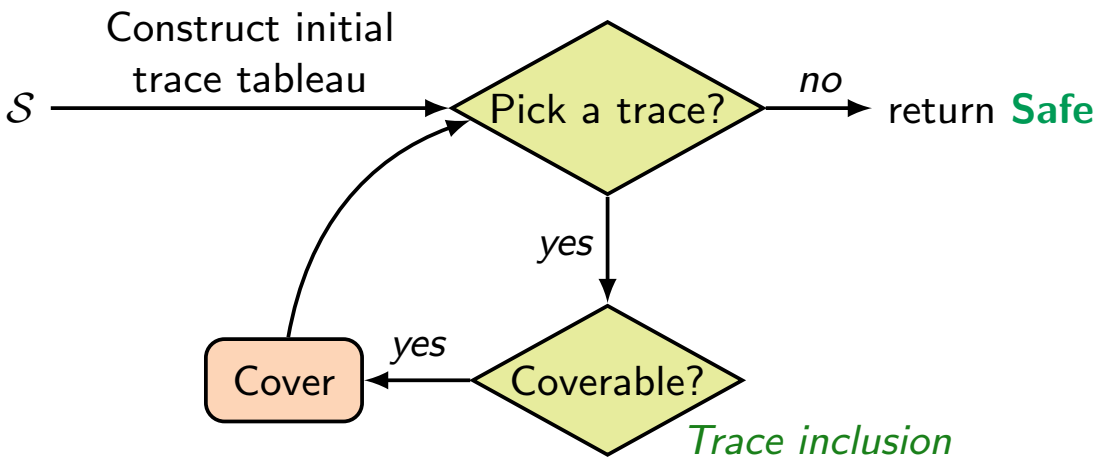
Reachability analysis algorithm

[K., Finkbeiner, CONCUR 2013]



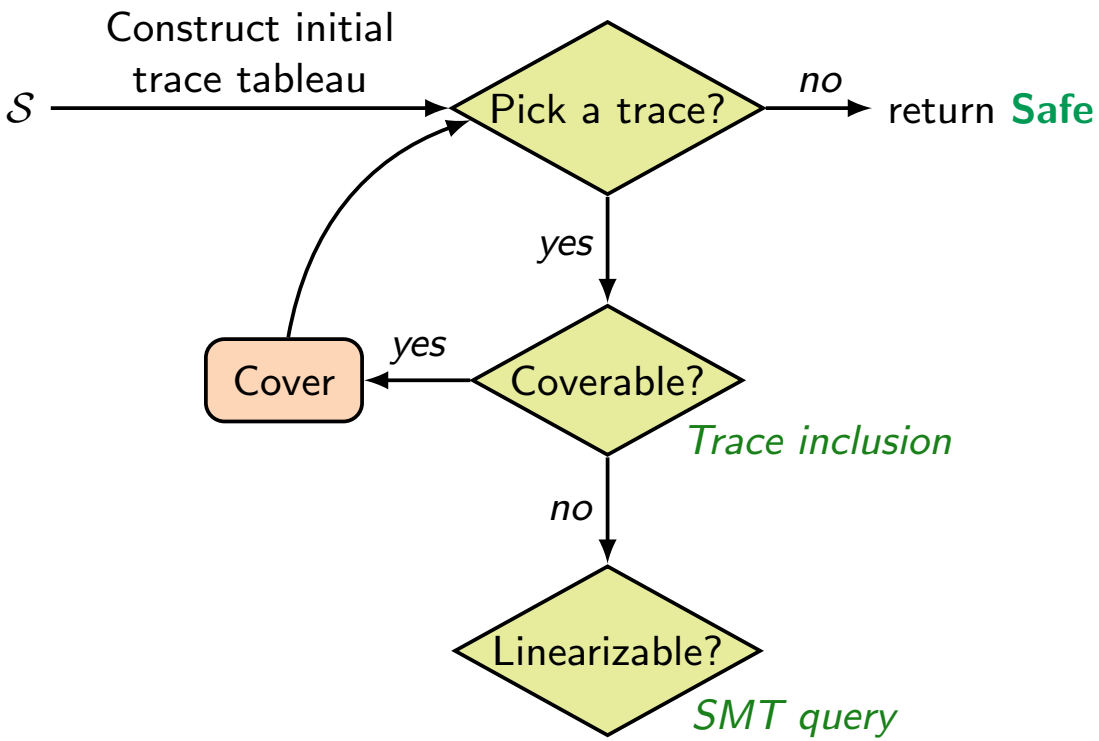
Reachability analysis algorithm

[K., Finkbeiner, CONCUR 2013]



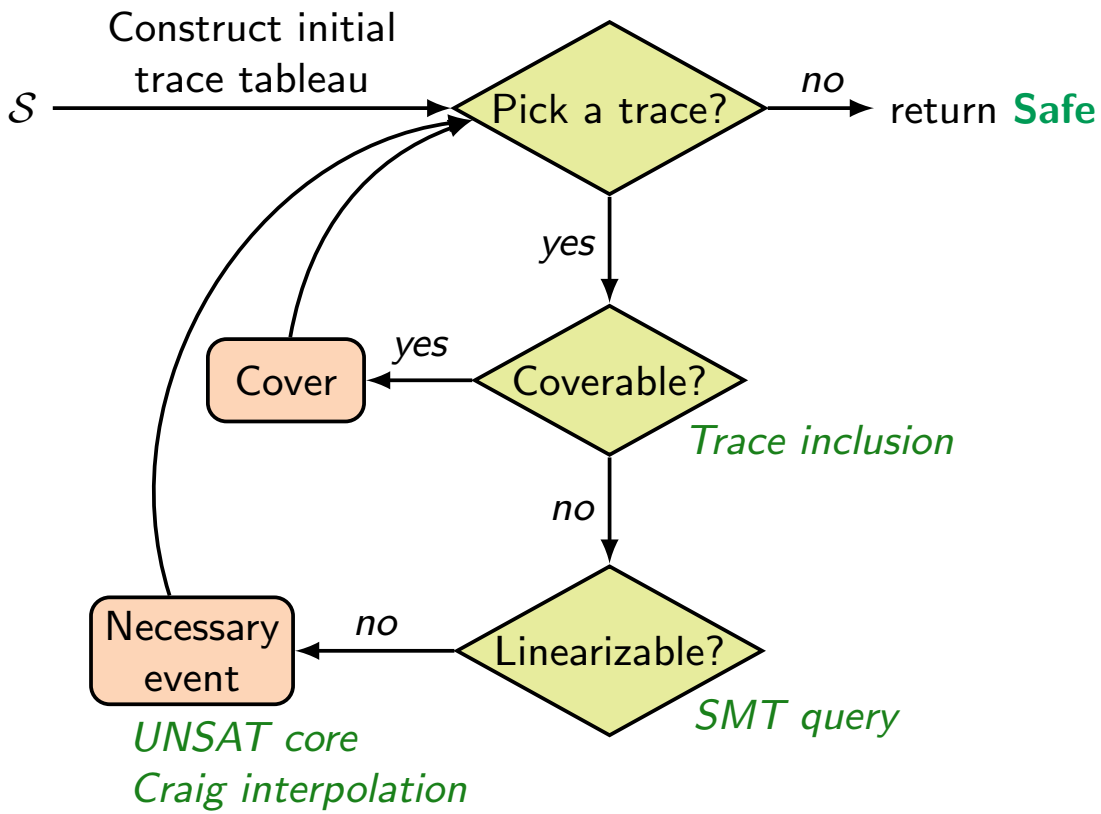
Reachability analysis algorithm

[K., Finkbeiner, CONCUR 2013]



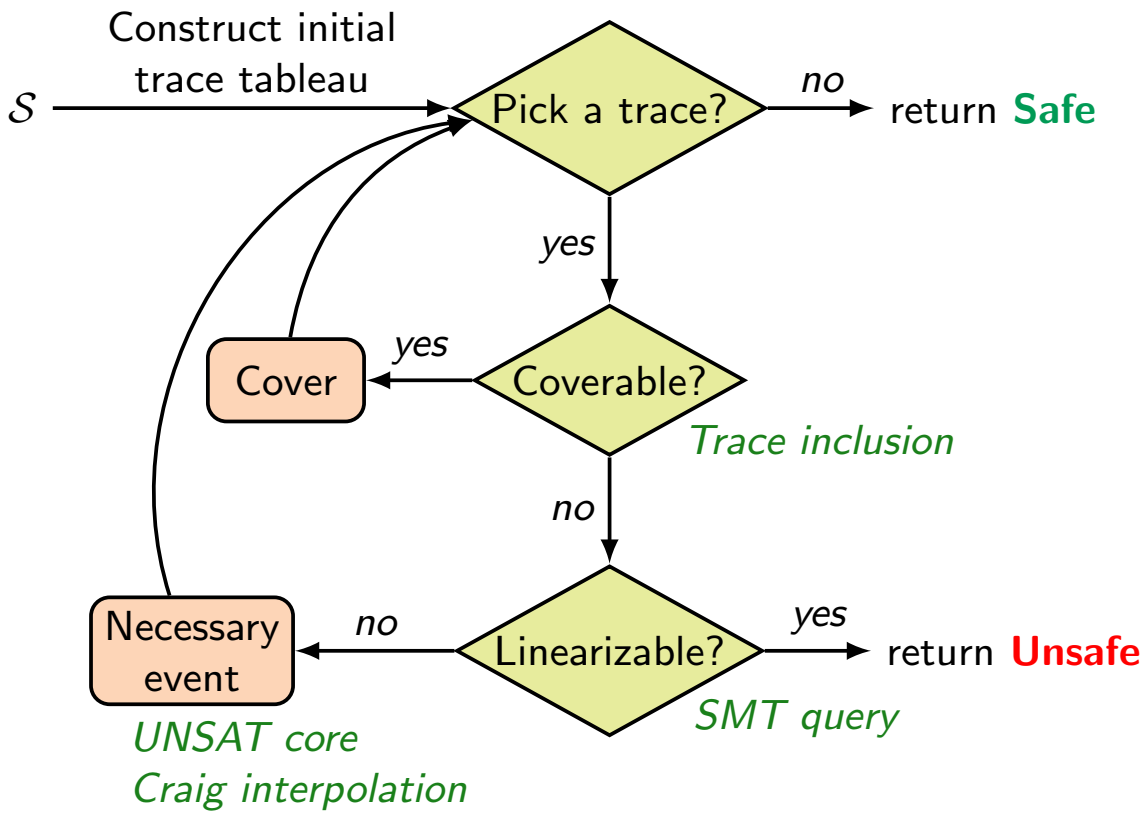
Reachability analysis algorithm

[K., Finkbeiner, CONCUR 2013]



Reachability analysis algorithm

[K., Finkbeiner, CONCUR 2013]



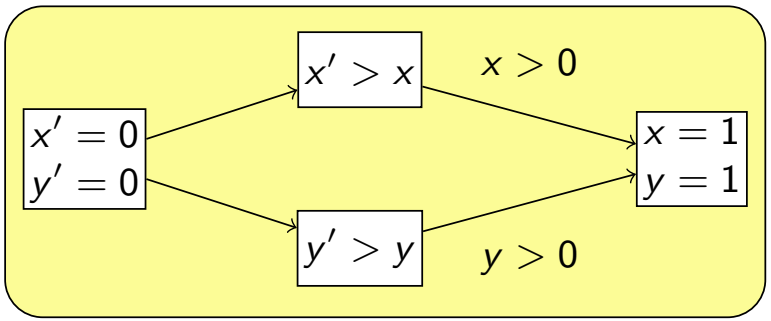
Trace inclusion

Finite traces

$A = \langle N, E, \nu, \eta \rangle \subseteq A' = \langle N', E', \nu', \eta' \rangle$ iff

- $\exists$ a pair of maps $\langle \lambda_N : N' \to N, \lambda_E : E' \to E \rangle$
- for all $n' \in N' \,.\, \nu(\lambda_N(n')) \implies \nu'(n')$
- for all $e' \in E' \,.\, \eta(\lambda_E(e')) \implies \eta'(e')$

$$A \subseteq A' \implies \mathcal{L}(A) \subseteq \mathcal{L}(A')$$



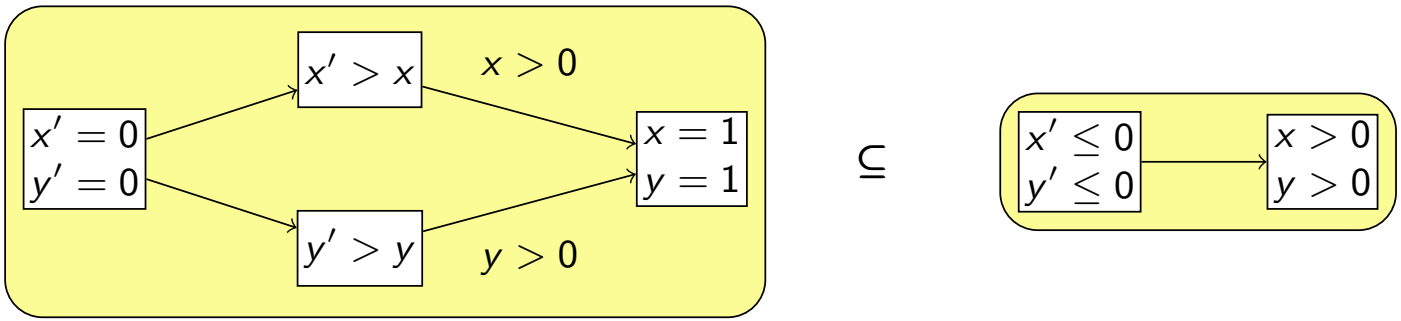
Trace inclusion

Finite traces

$A = \langle N, E, \nu, \eta \rangle \subseteq A' = \langle N', E', \nu', \eta' \rangle$ iff

- $\exists$ a pair of maps $\langle \lambda_N : N' \to N, \lambda_E : E' \to E \rangle$
- for all $n' \in N' \,.\, \nu(\lambda_N(n')) \implies \nu'(n')$
- for all $e' \in E' \,.\, \eta(\lambda_E(e')) \implies \eta'(e')$

$A \subseteq A' \implies \mathcal{L}(A) \subseteq \mathcal{L}(A')$



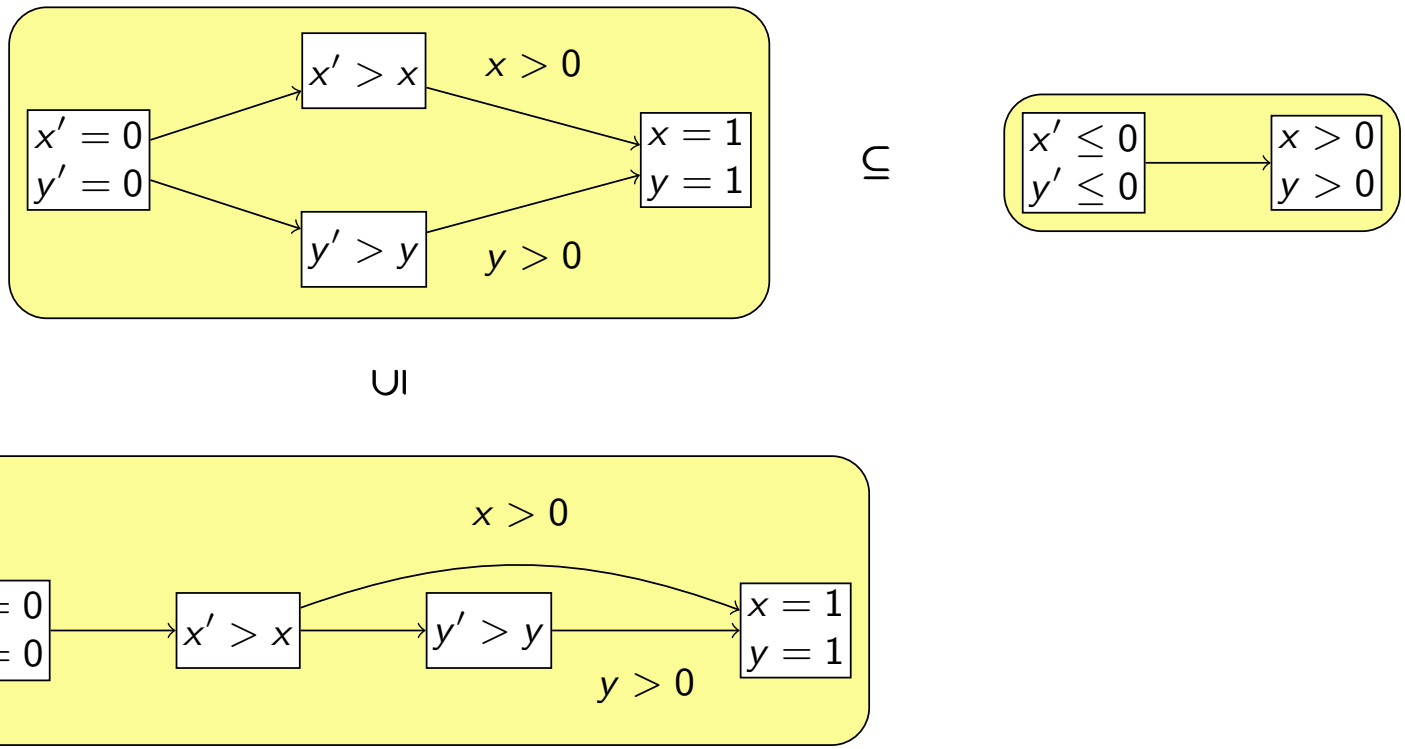
Trace inclusion

Finite traces

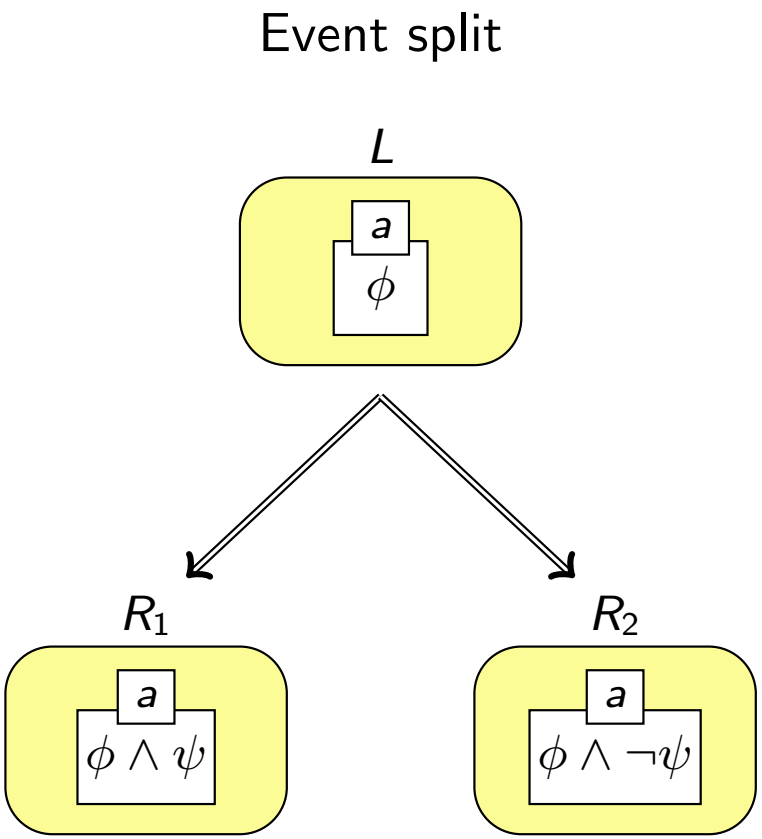
$A = \langle N, E, \nu, \eta \rangle \subseteq A' = \langle N', E', \nu', \eta' \rangle$ iff

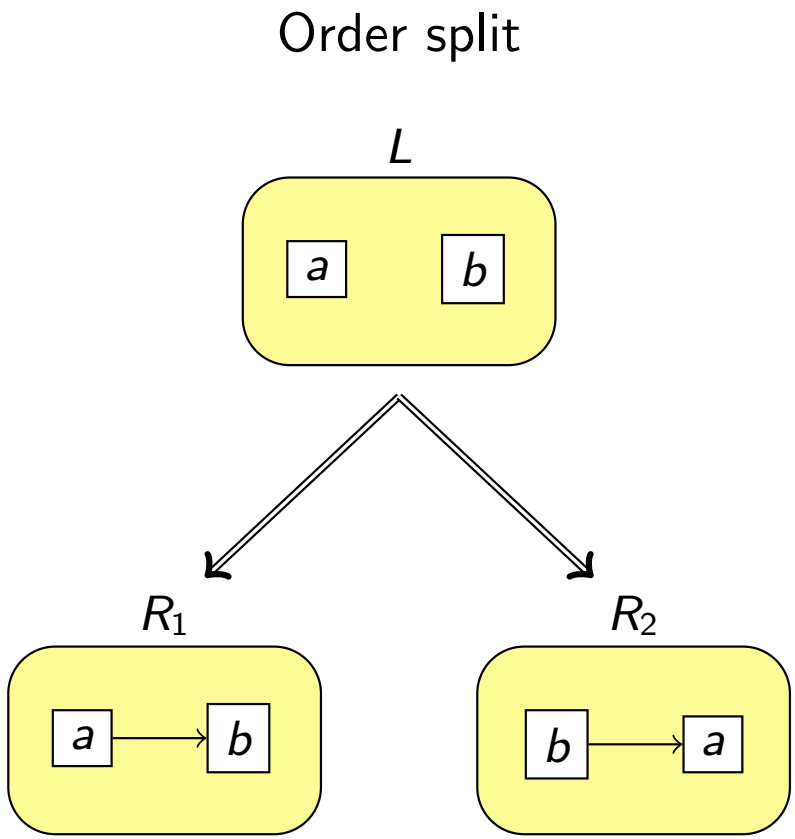
- $\exists$ a pair of maps $\langle \lambda_N : N' \to N, \lambda_E : E' \to E \rangle$
- for all $n' \in N' \,.\, \nu(\lambda_N(n')) \implies \nu'(n')$
- for all $e' \in E' \,.\, \eta(\lambda_E(e')) \implies \eta'(e')$

$A \subseteq A' \implies \mathcal{L}(A) \subseteq \mathcal{L}(A')$



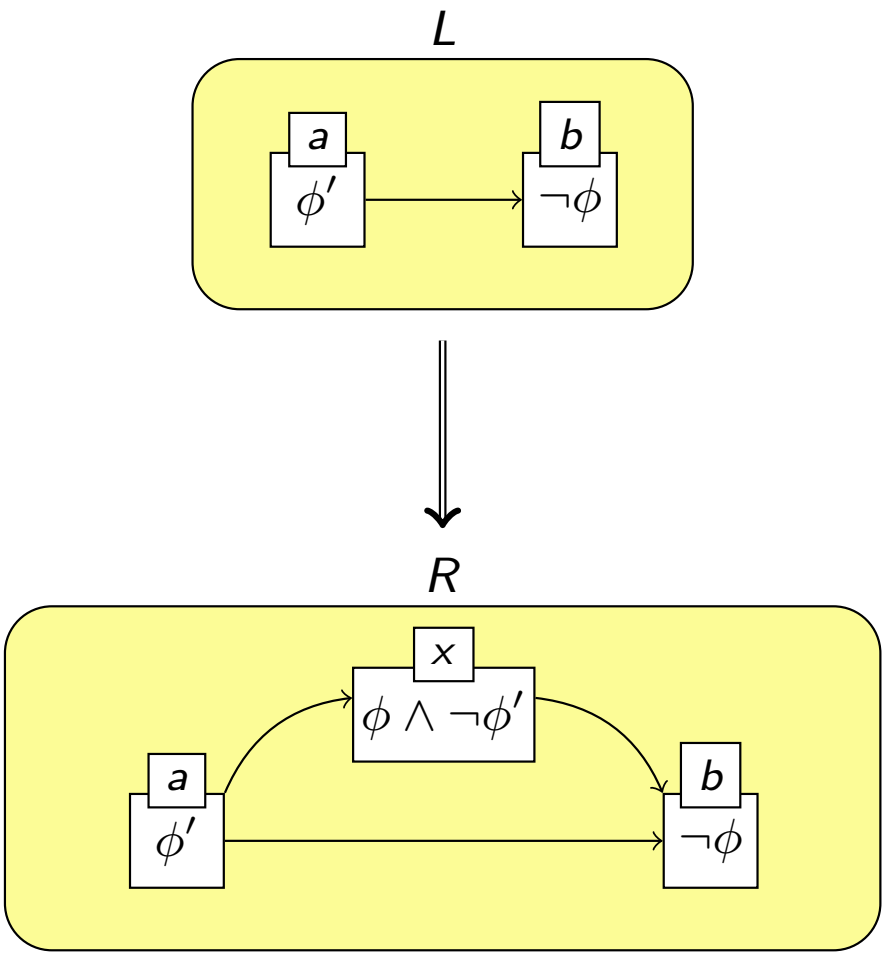
Proof rules: finite traces

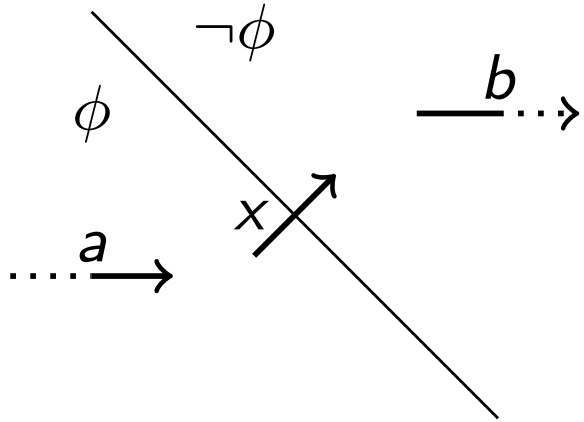




Proof rules: finite traces

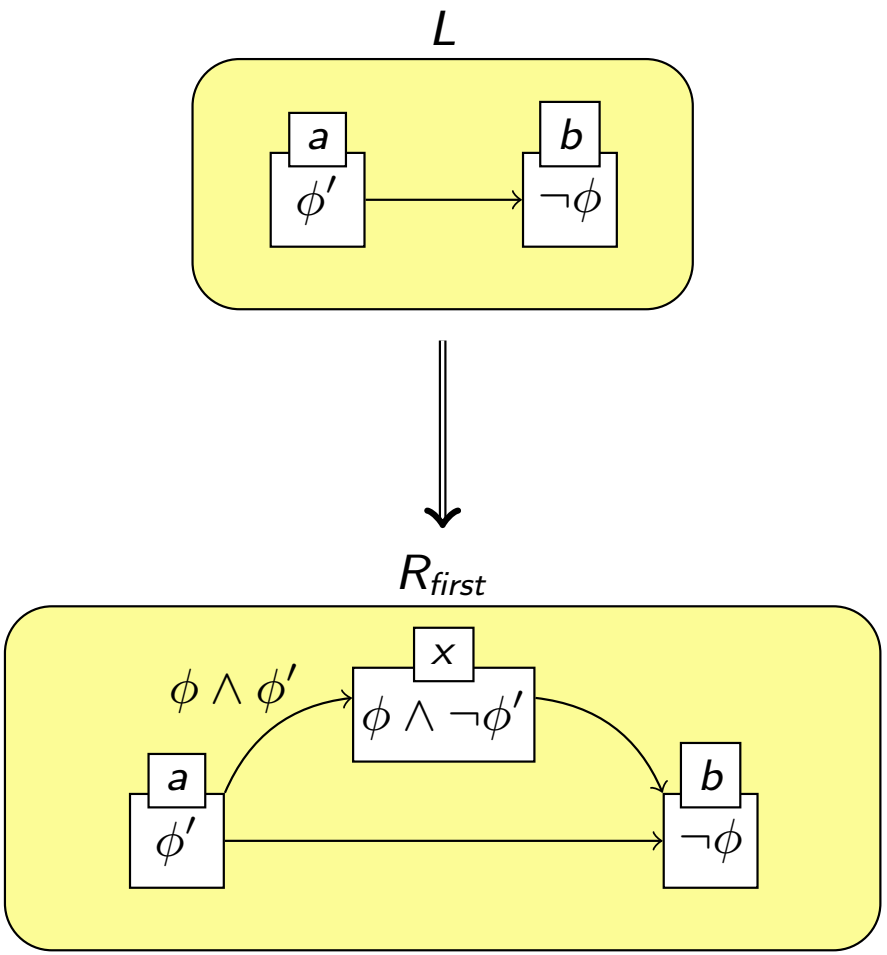
Necessary event

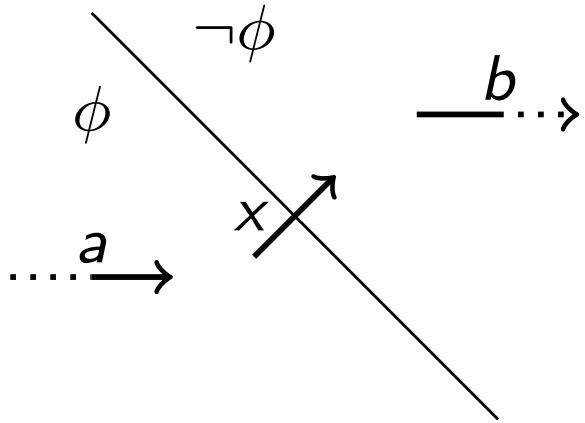




Proof rules: finite traces

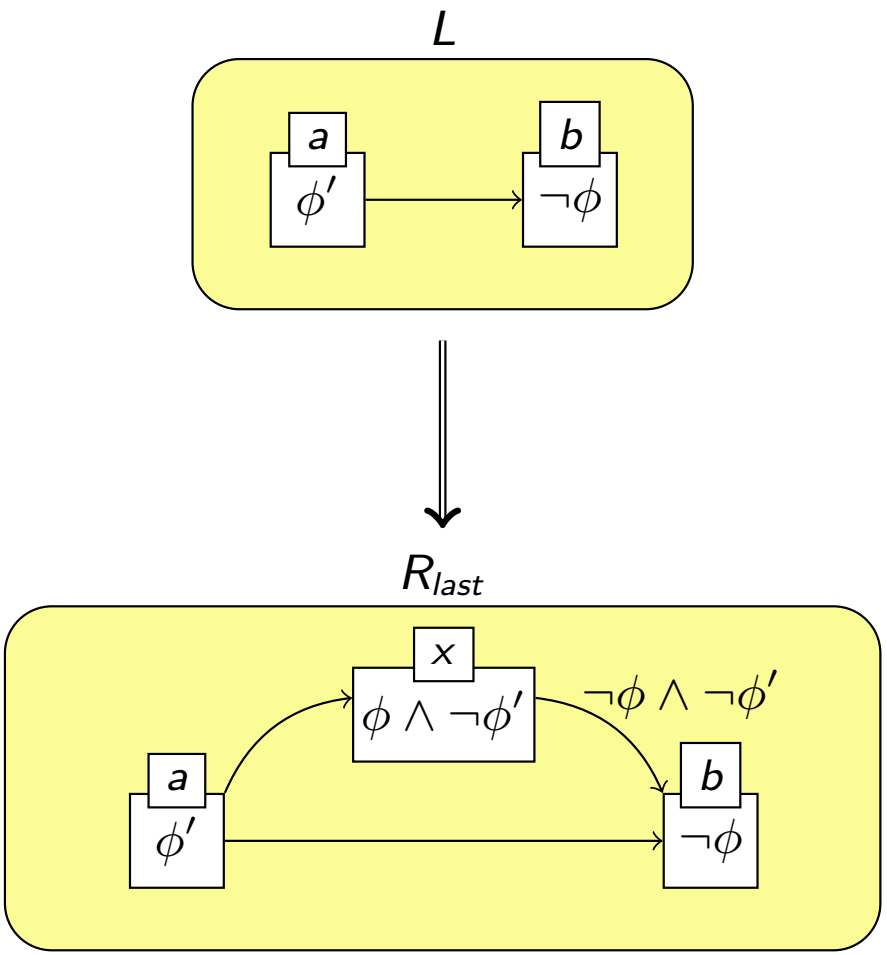
Necessary event

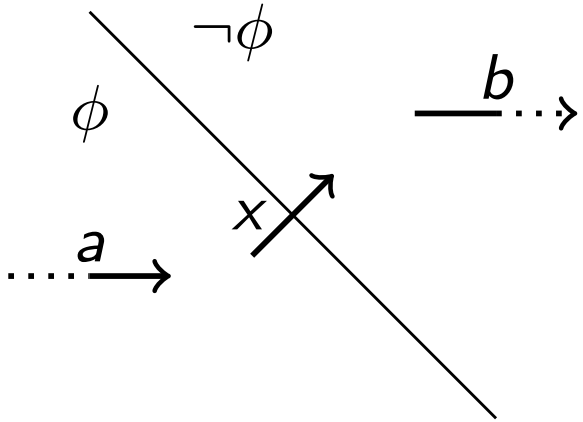




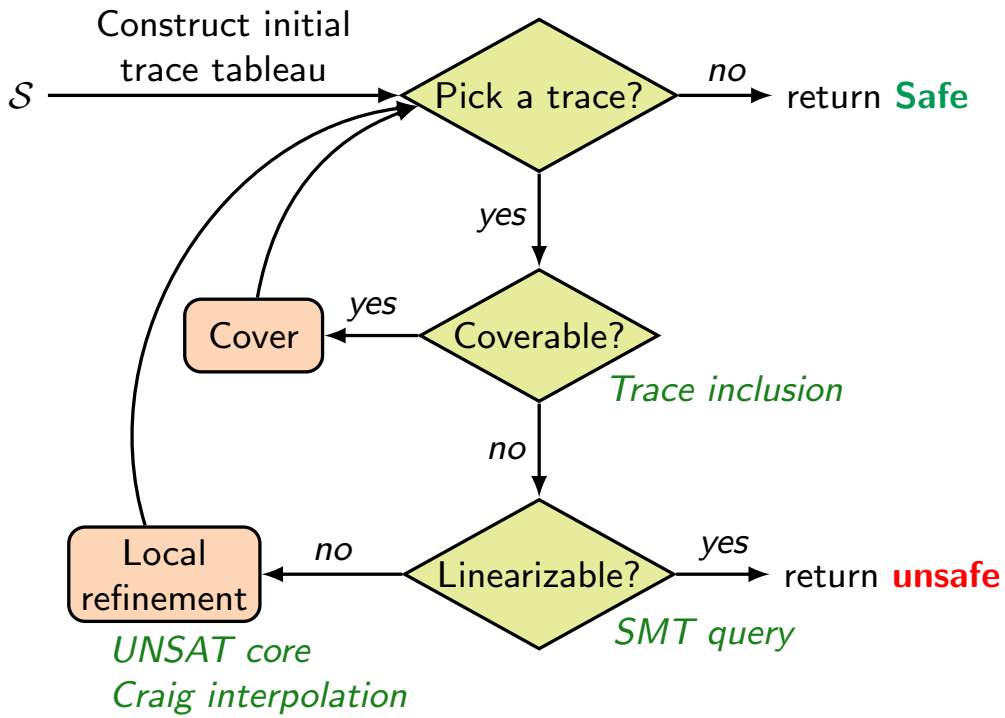
Proof rules: finite traces

Necessary event

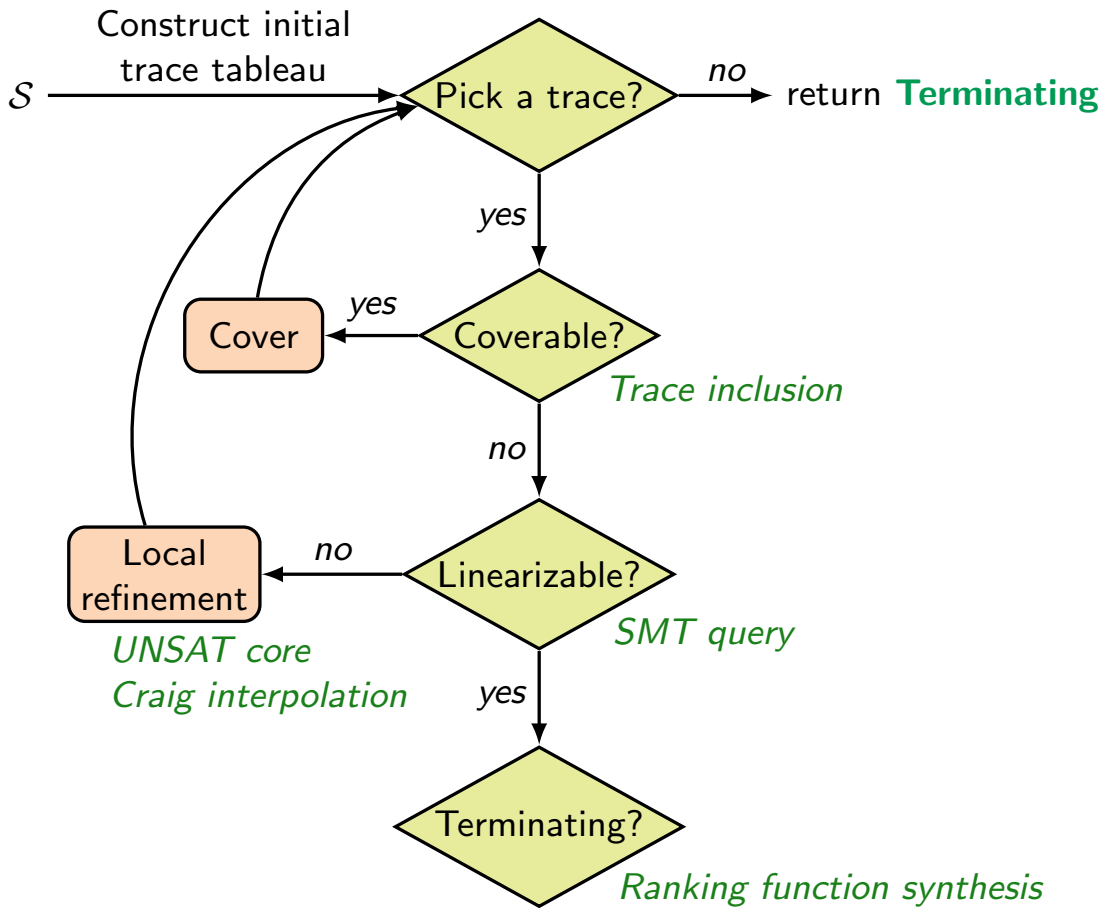




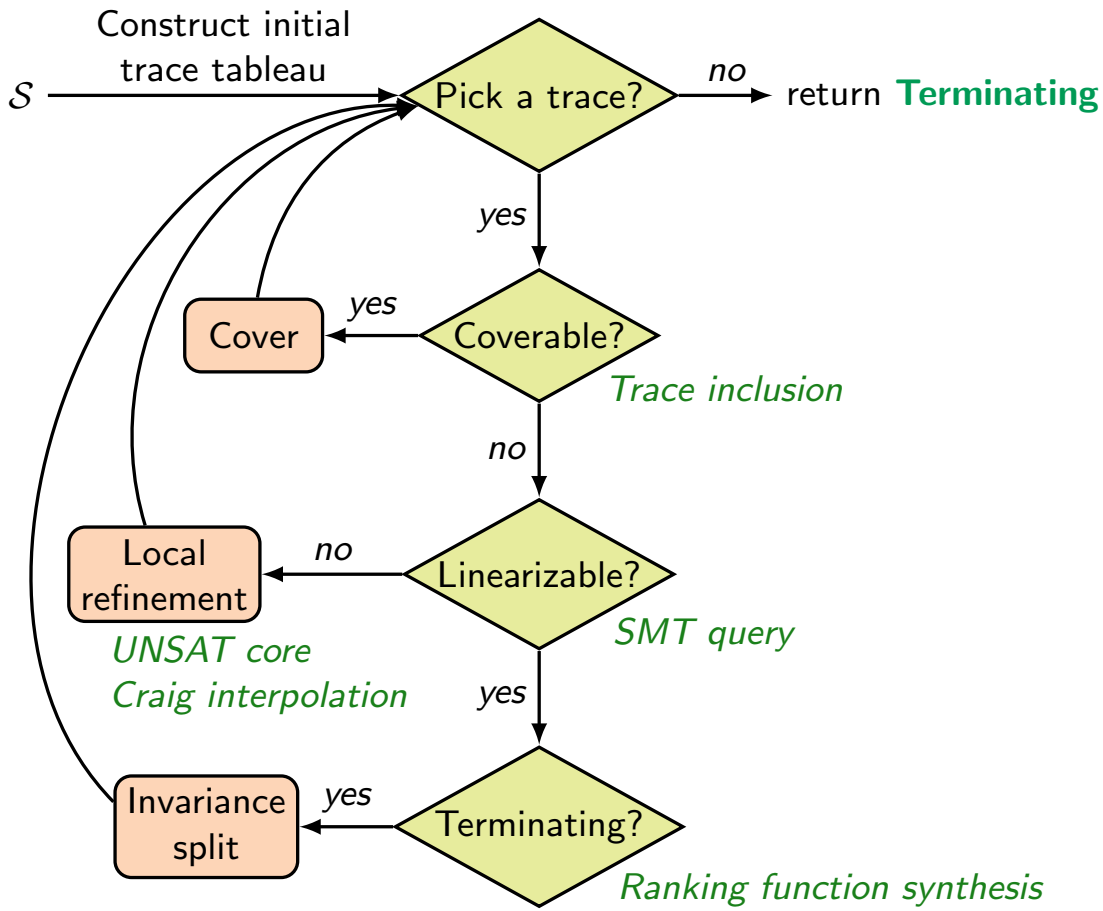
Termination analysis algorithm



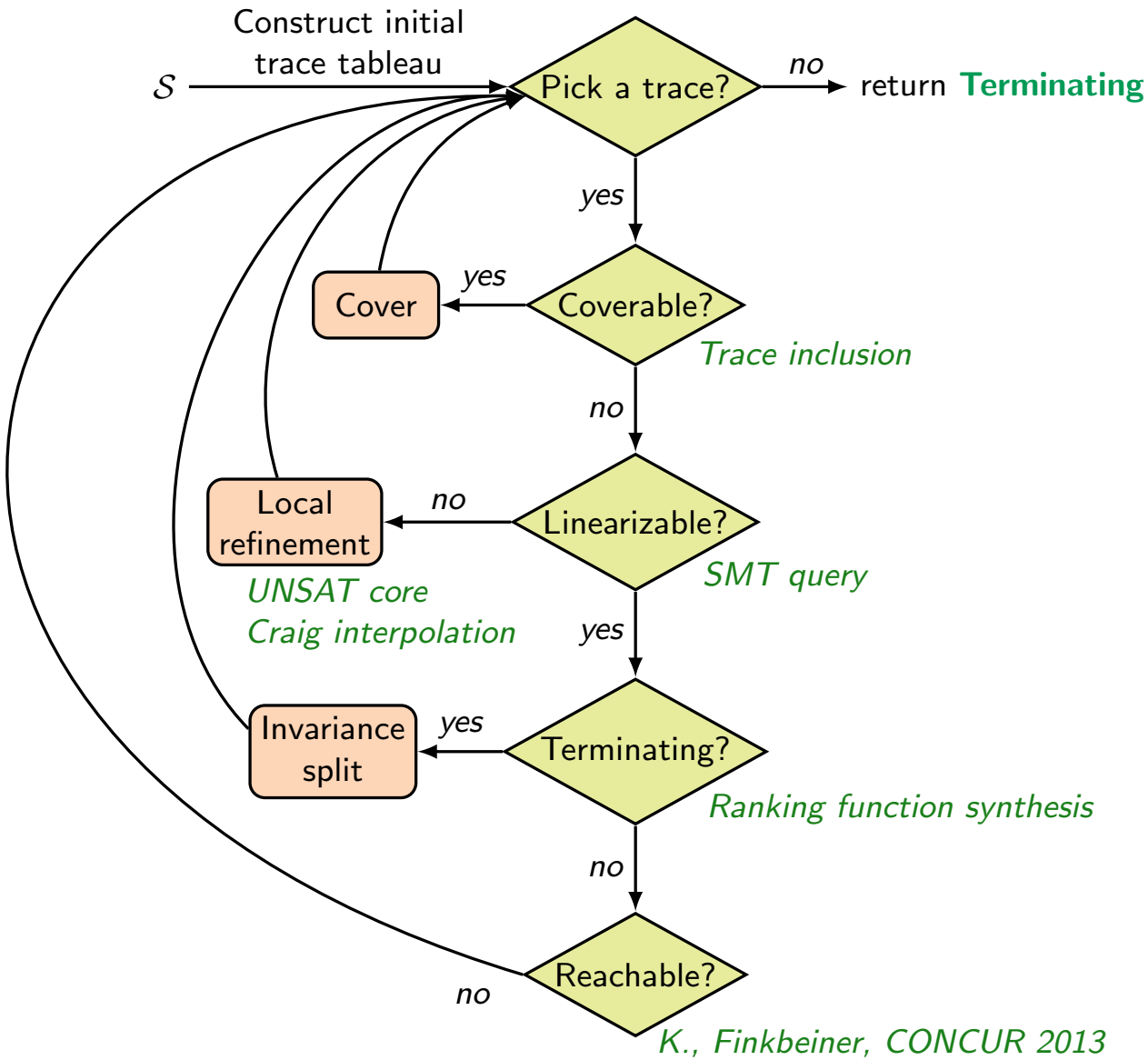
Termination analysis algorithm



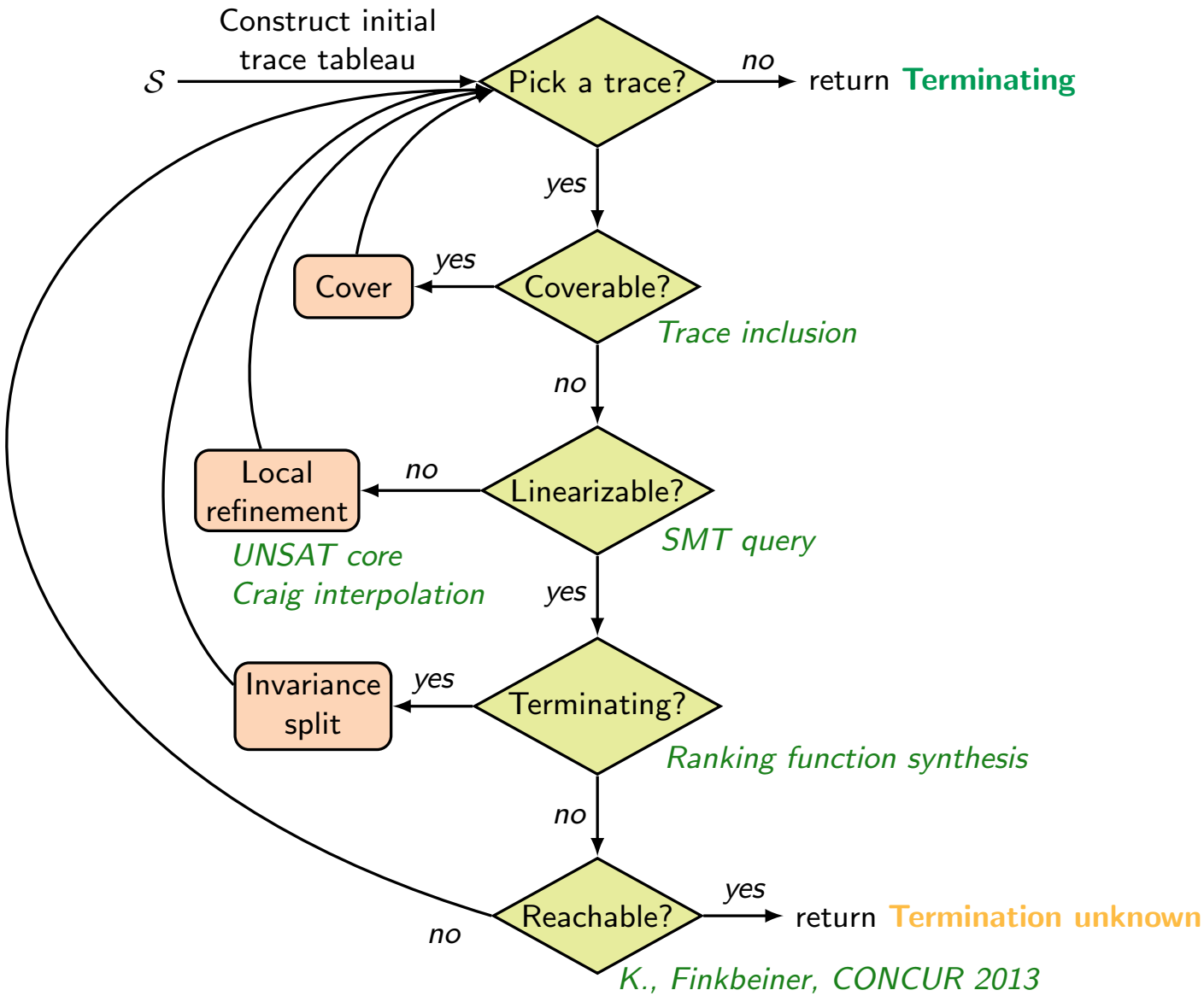
Termination analysis algorithm



Termination analysis algorithm



Termination analysis algorithm



Infinite concurrent traces

- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x}^{+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x}^{-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$

Infinite concurrent traces

- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x}^{+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x}^{-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

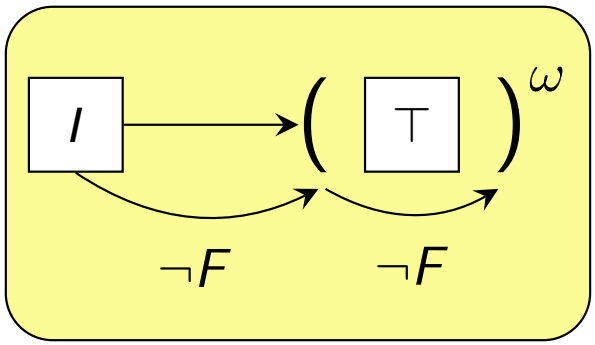
Infinite concurrent traces

Infinite trace $\mathcal{I} = \langle \mathcal{A}_s, \mathcal{A}_c, \phi_s, \phi_c \rangle$

- $\mathcal{A}_s, \mathcal{A}_C$: *stem* and *cycle* traces
- ϕ_s, $\phi_c \in \Phi(V \cup V')$

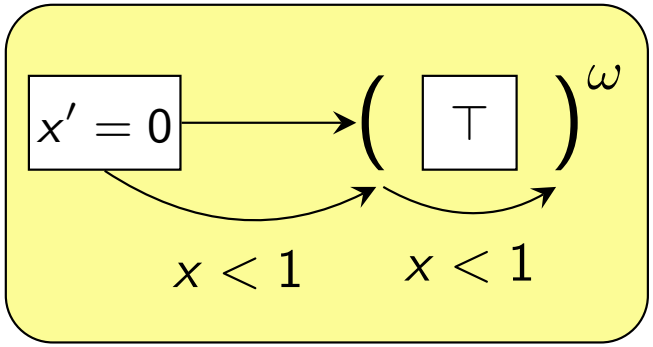
- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x^-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

Infinite concurrent traces



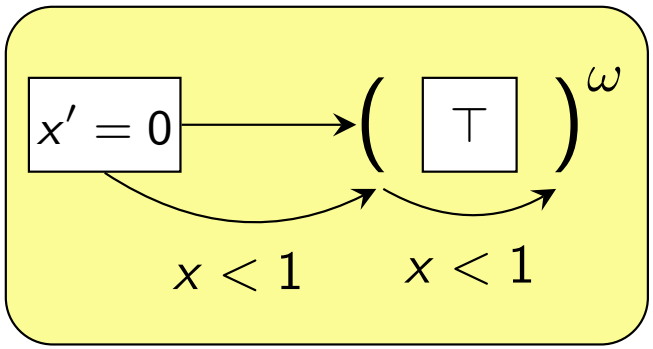
- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x^-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

Infinite concurrent traces



- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x}^{+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x}^{-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

Infinite concurrent traces

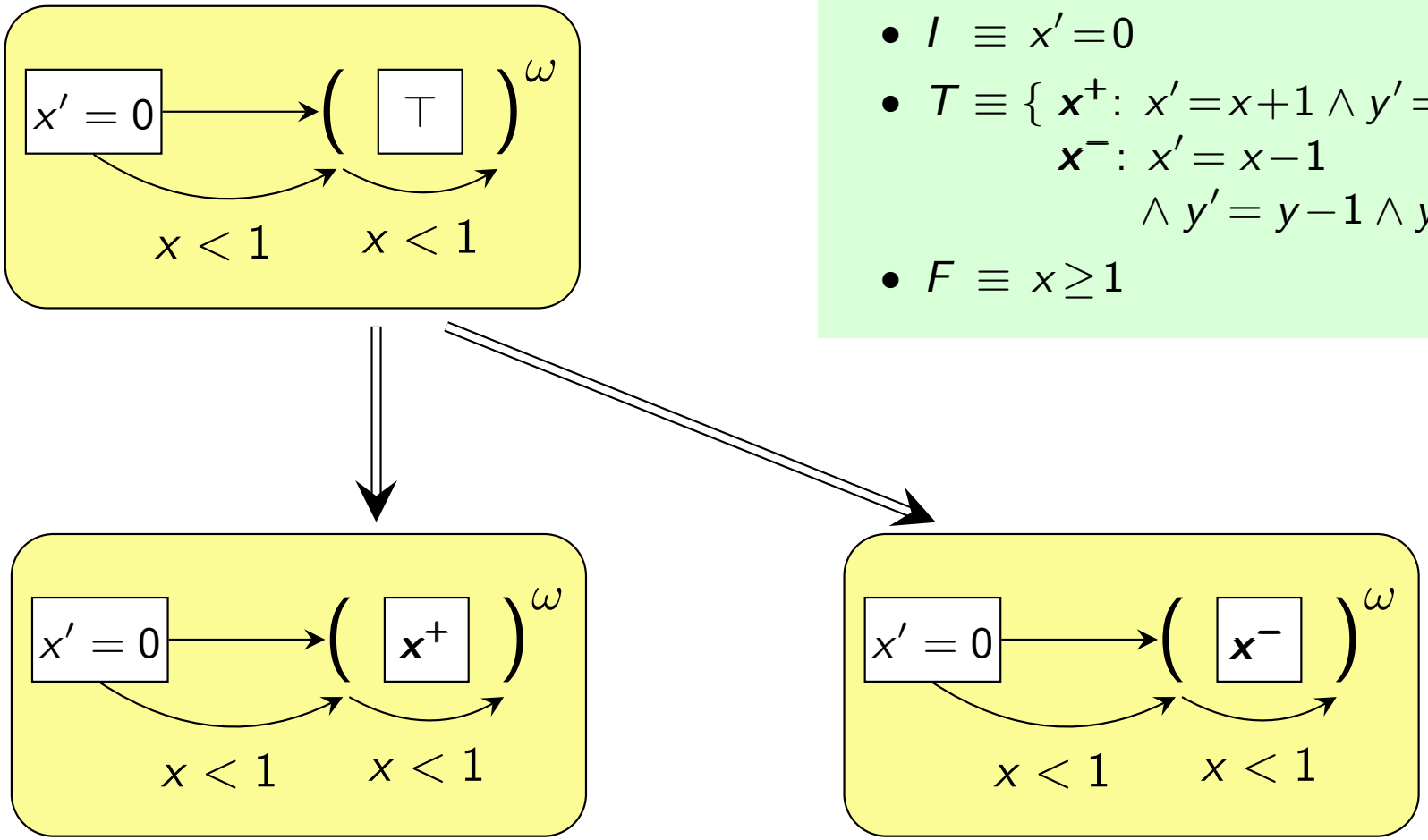


- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x}^{+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x}^{-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

Language of an infinite concurrent trace

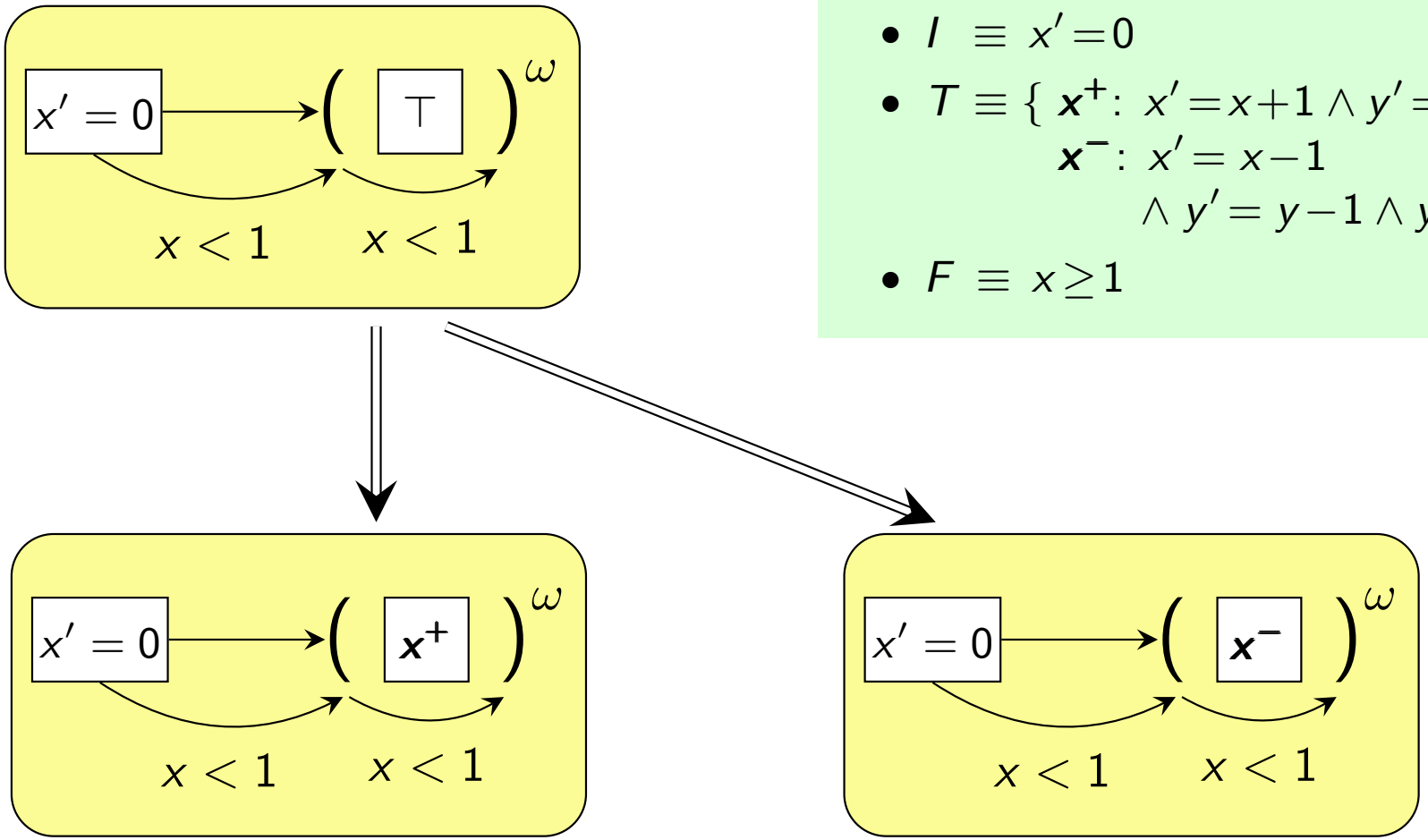
A set of system traces such that the *stem* can be embedded once, and the *cycle* infinitely often after the stem, into an infinite subsequence of a system trace, *respecting constraints*

Infinite concurrent traces



- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x^-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

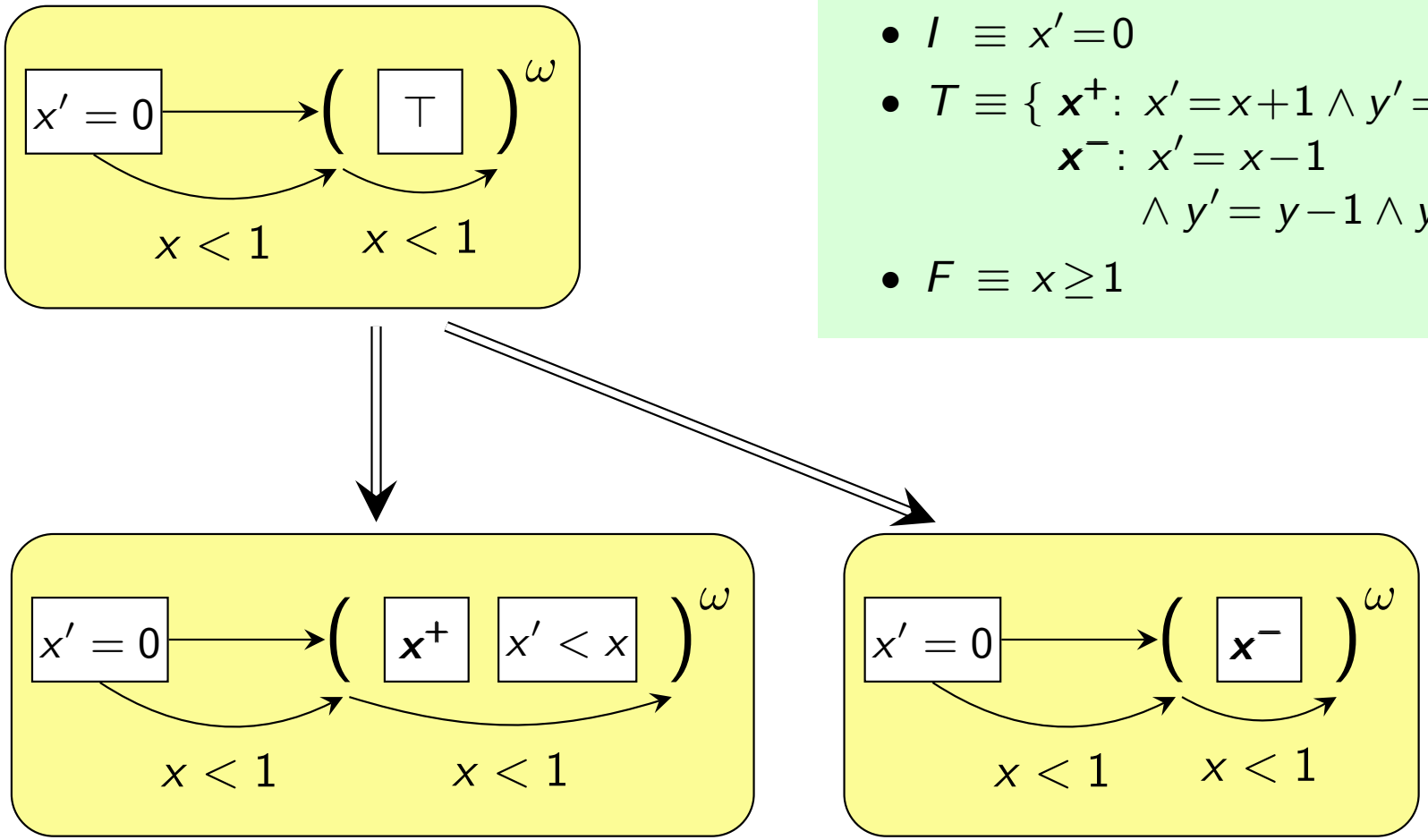
Infinite concurrent traces



- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x^+}\colon\ x' = x+1 \wedge y' = y$
  $\boldsymbol{x^-}\colon\ x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

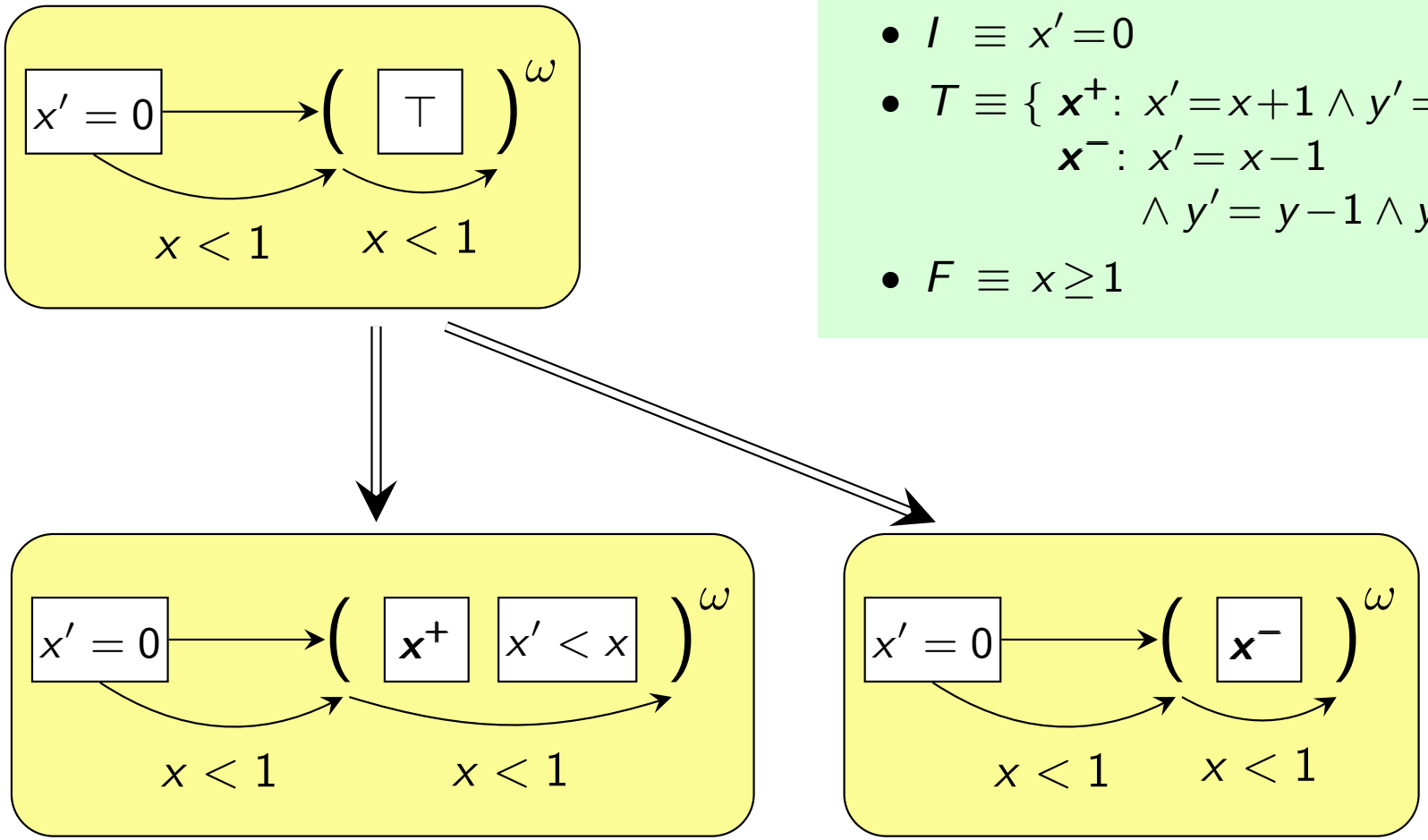
Terminating: $1 - x$

Infinite concurrent traces



- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x}^+ \colon x' = x+1 \wedge y' = y$
  $\boldsymbol{x}^- \colon x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

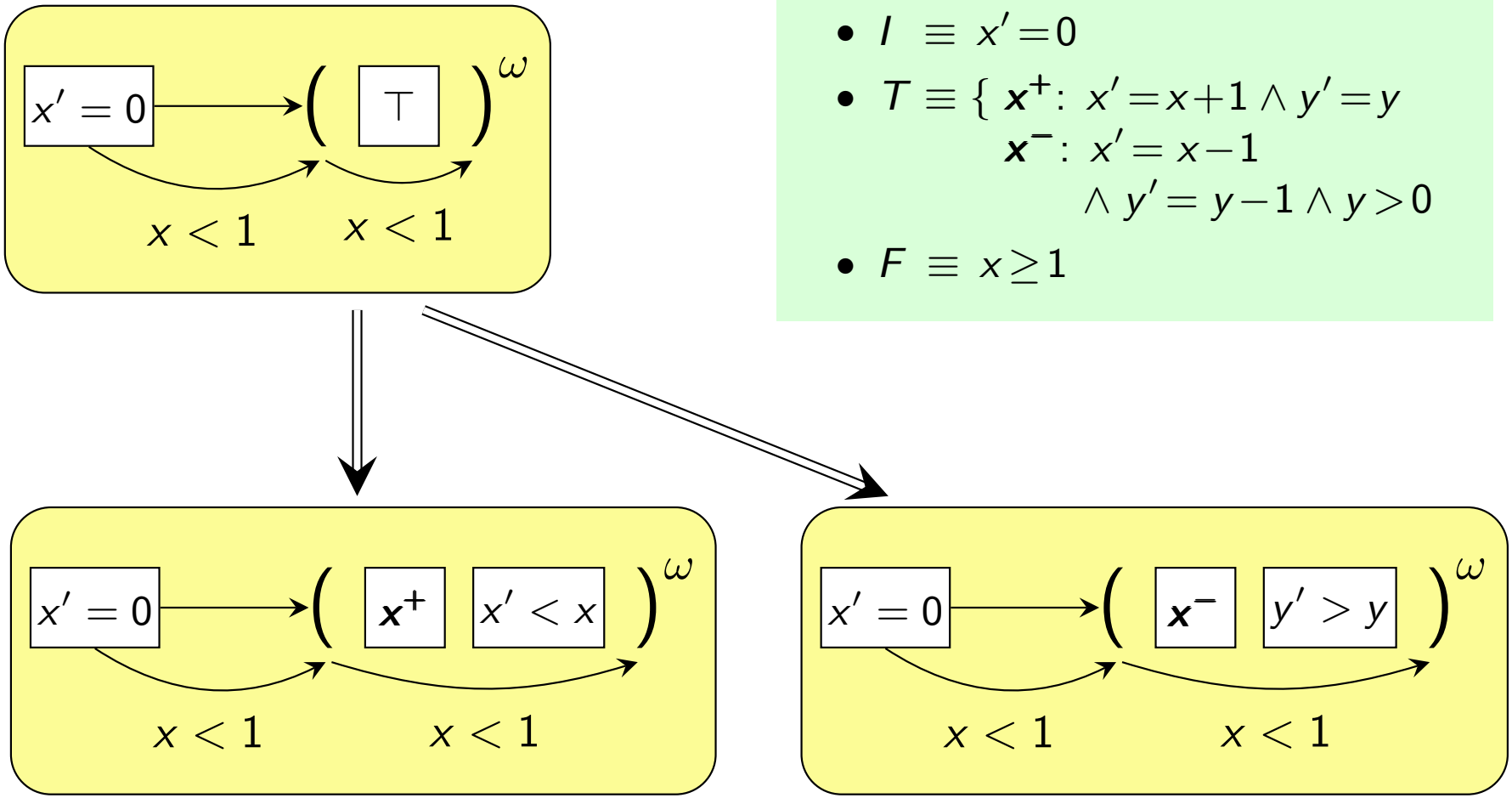
Infinite concurrent traces



- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x}^+ \colon x' = x+1 \wedge y' = y$
  $\boldsymbol{x}^- \colon x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

Terminating: y

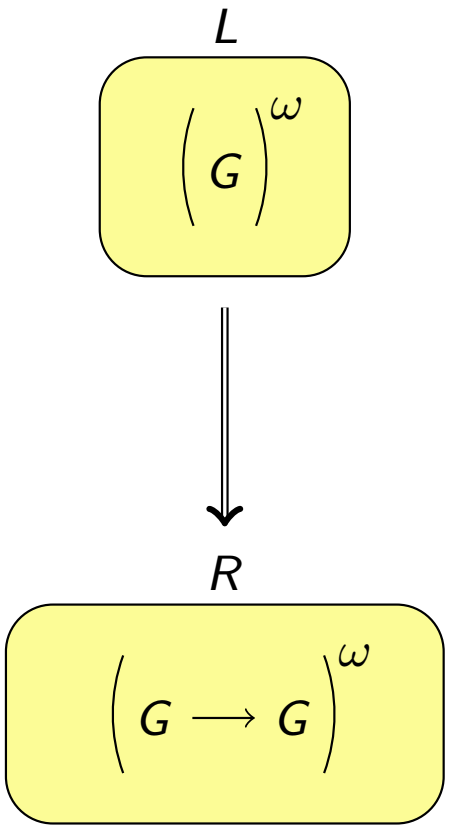
Infinite concurrent traces



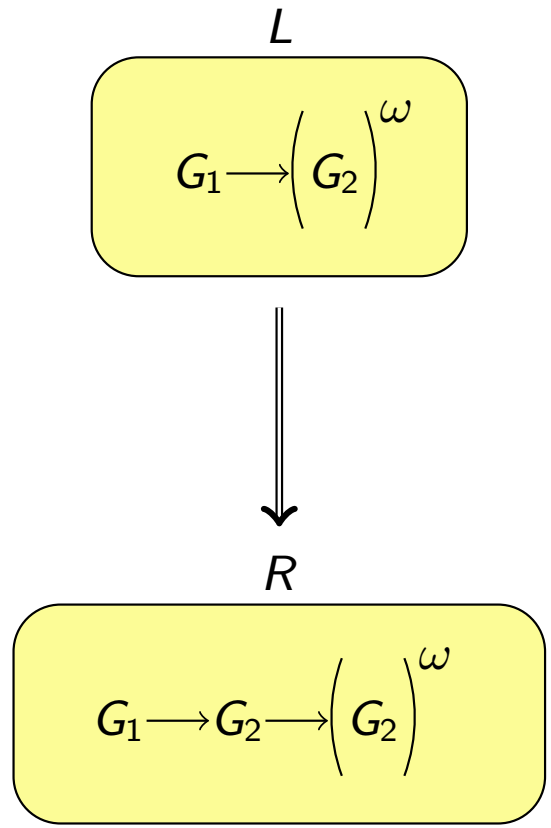
- $I \equiv x' = 0$
- $T \equiv \{\ \boldsymbol{x}^+ \colon x' = x+1 \wedge y' = y$
  $\boldsymbol{x}^- \colon x' = x-1$
  $\wedge\ y' = y-1 \wedge y > 0$
- $F \equiv x \geq 1$

Proof rules: infinite traces

Unrolling

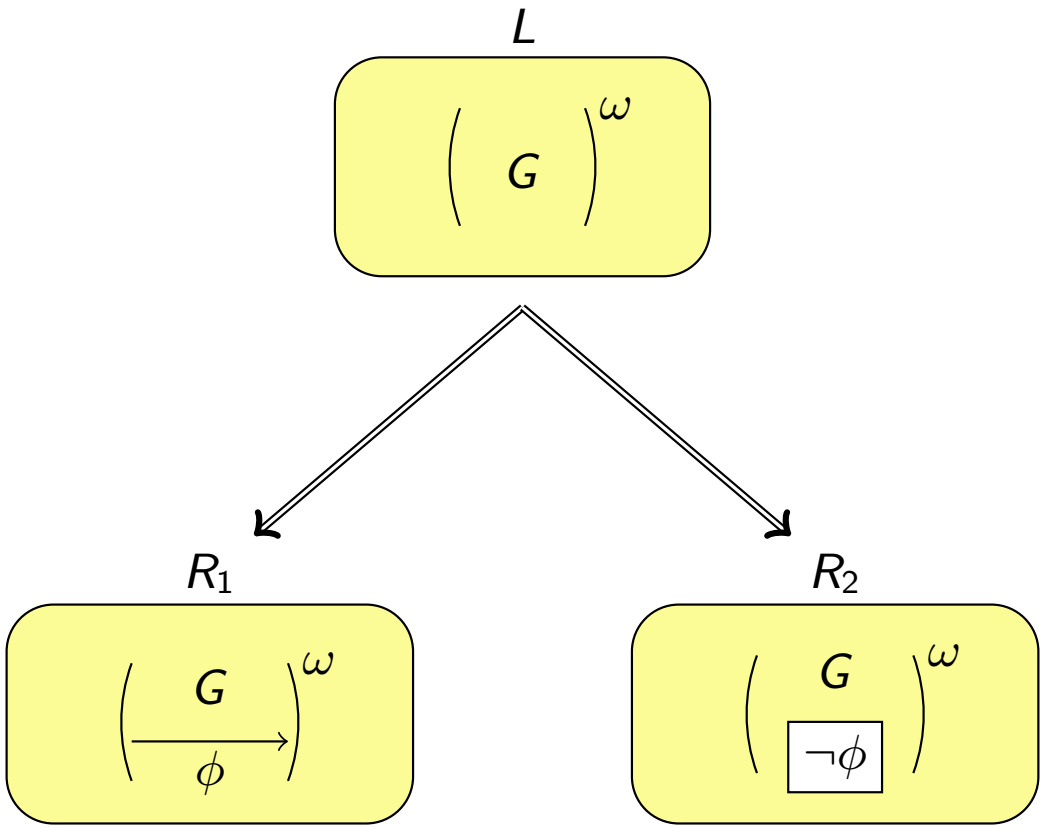


Cycle-to-stem



Proof rules: infinite traces

Invariance split



Soundness and completeness

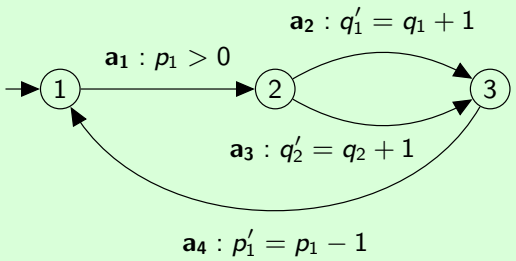
Theorem (Soundness)

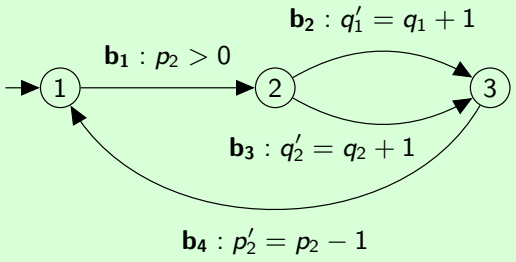
If there exists a correct and complete causal trace tableau for a transition system $\mathcal{S}$, then $\mathcal{S}$ is terminating.

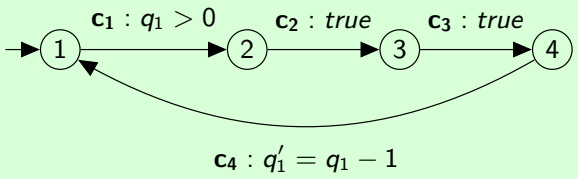
Theorem (Relative completeness)

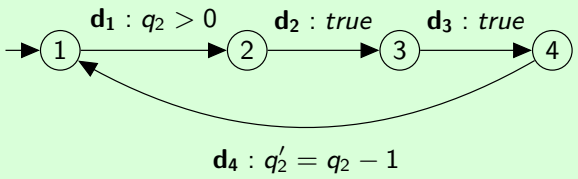
If a transition system $\mathcal{S}$ is terminating, then a correct and complete causal trace tableau for $\mathcal{S}$ can be constructed, provided that all necessary first-order formulas are given.

Example: Producer-Consumer

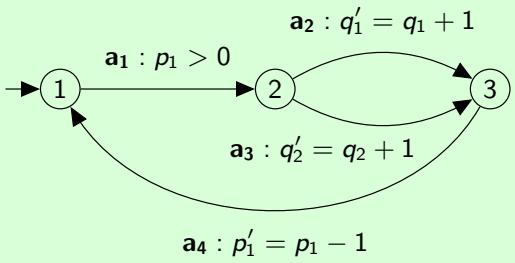


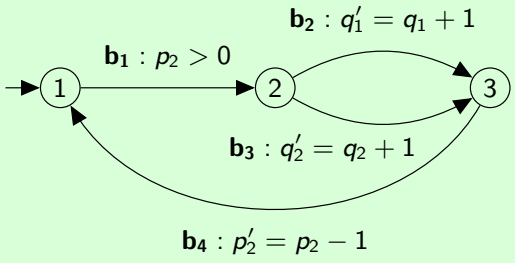


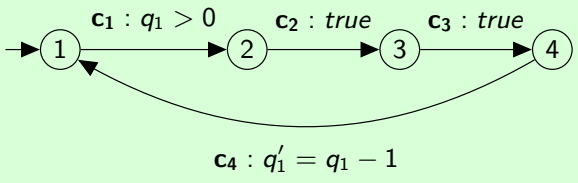


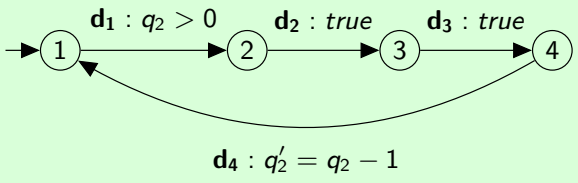


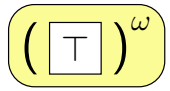
Example: Producer-Consumer



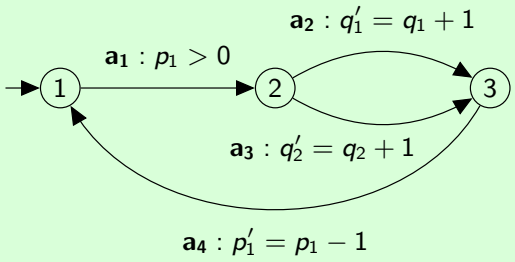


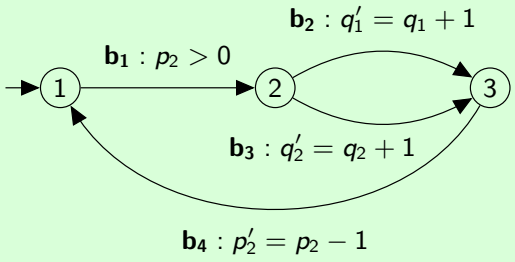


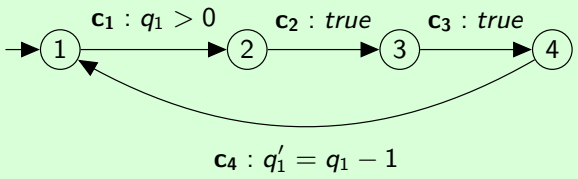


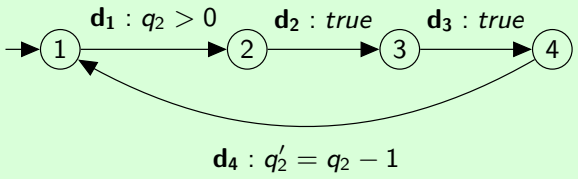


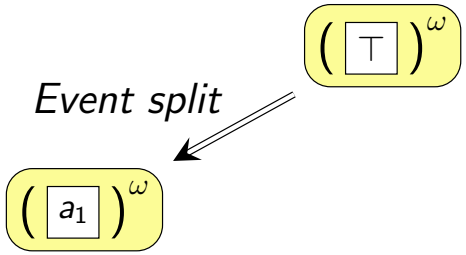
Example: Producer-Consumer



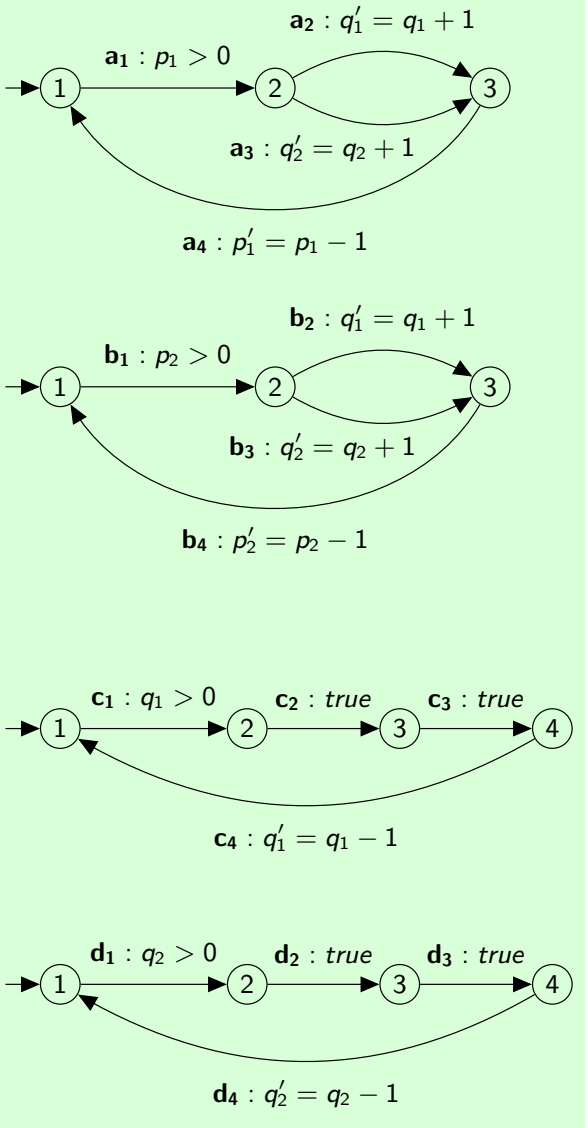


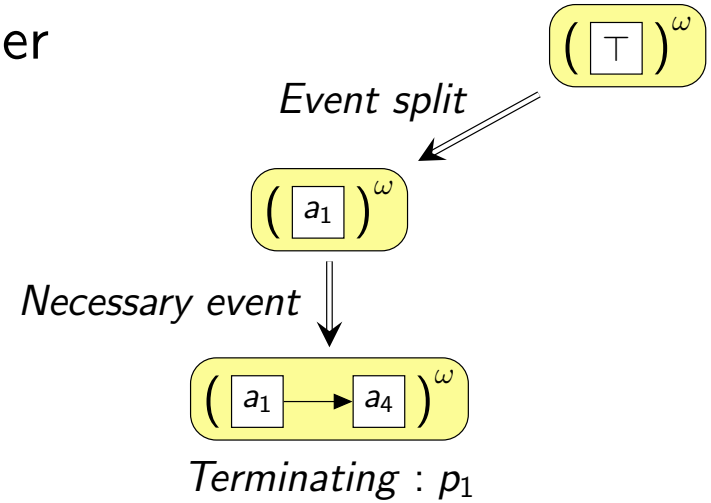




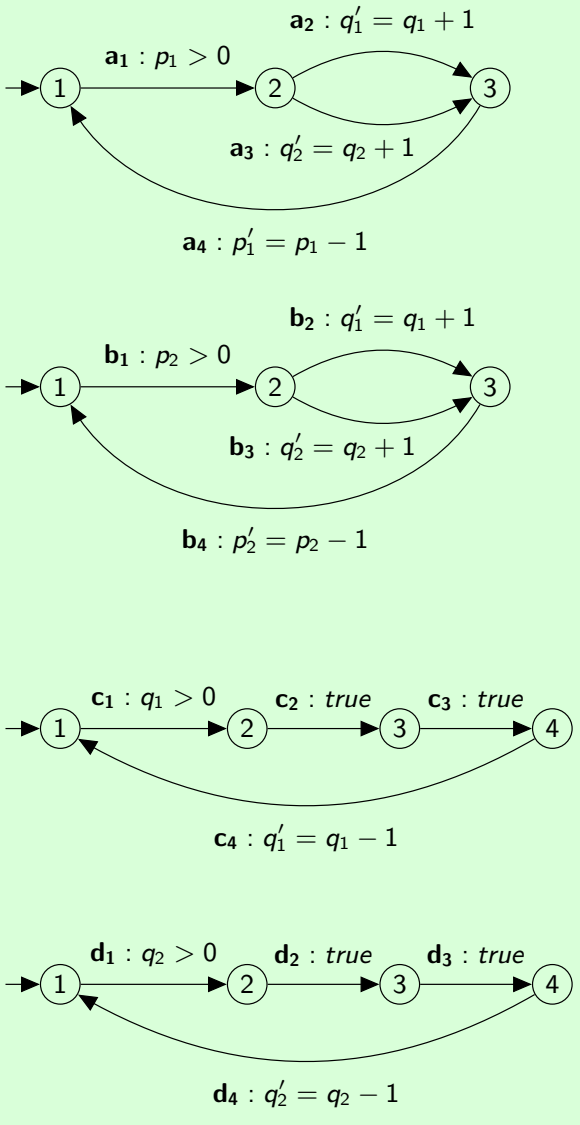


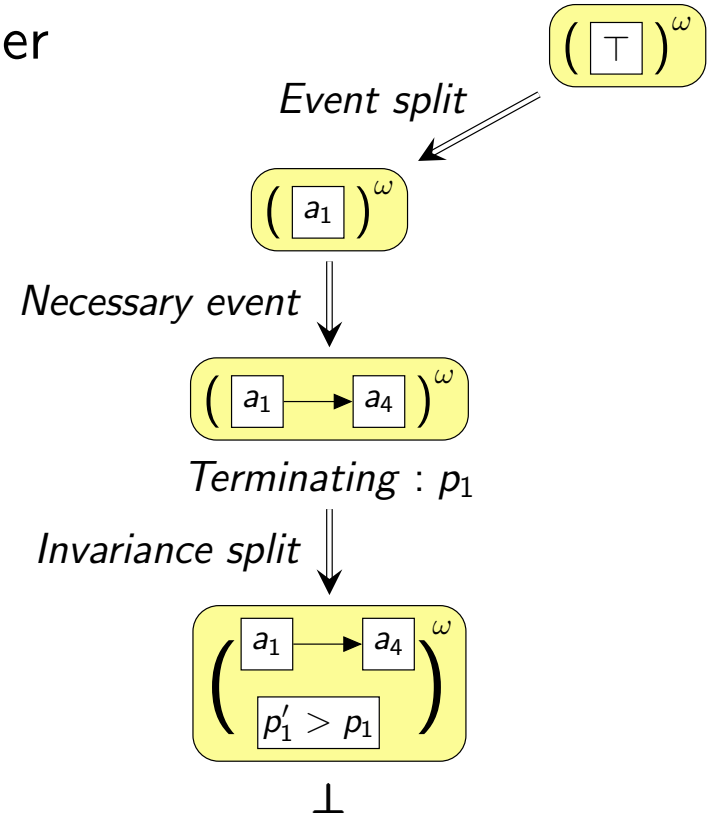
Example: Producer-Consumer



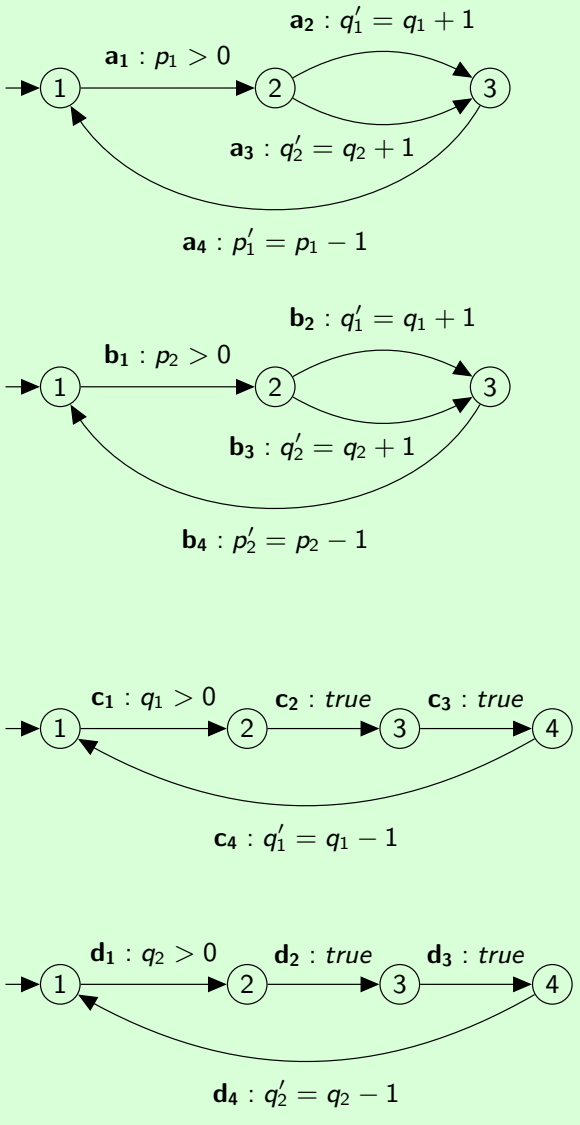


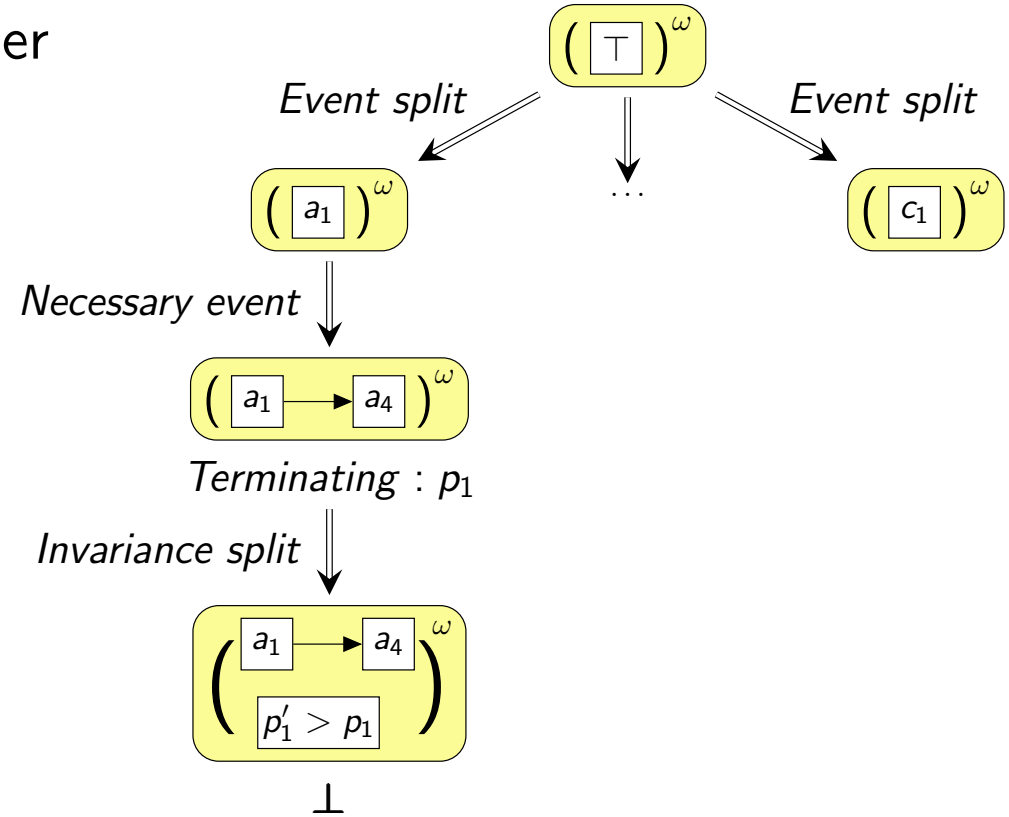
Example: Producer-Consumer



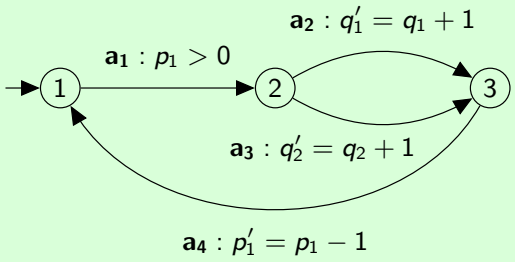


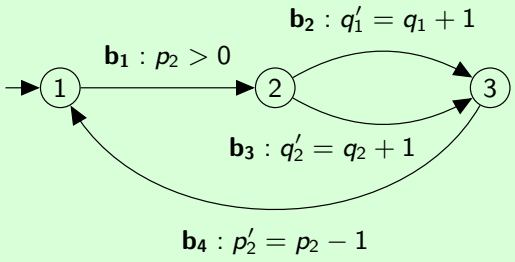
Example: Producer-Consumer

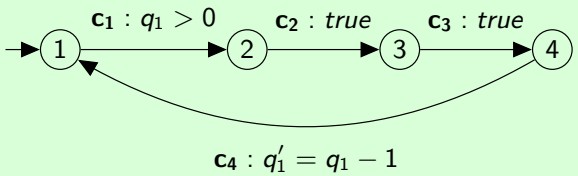


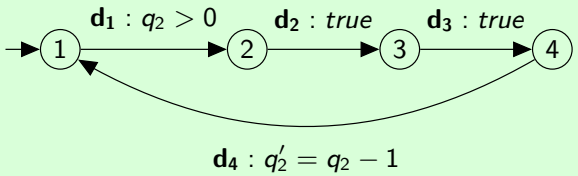


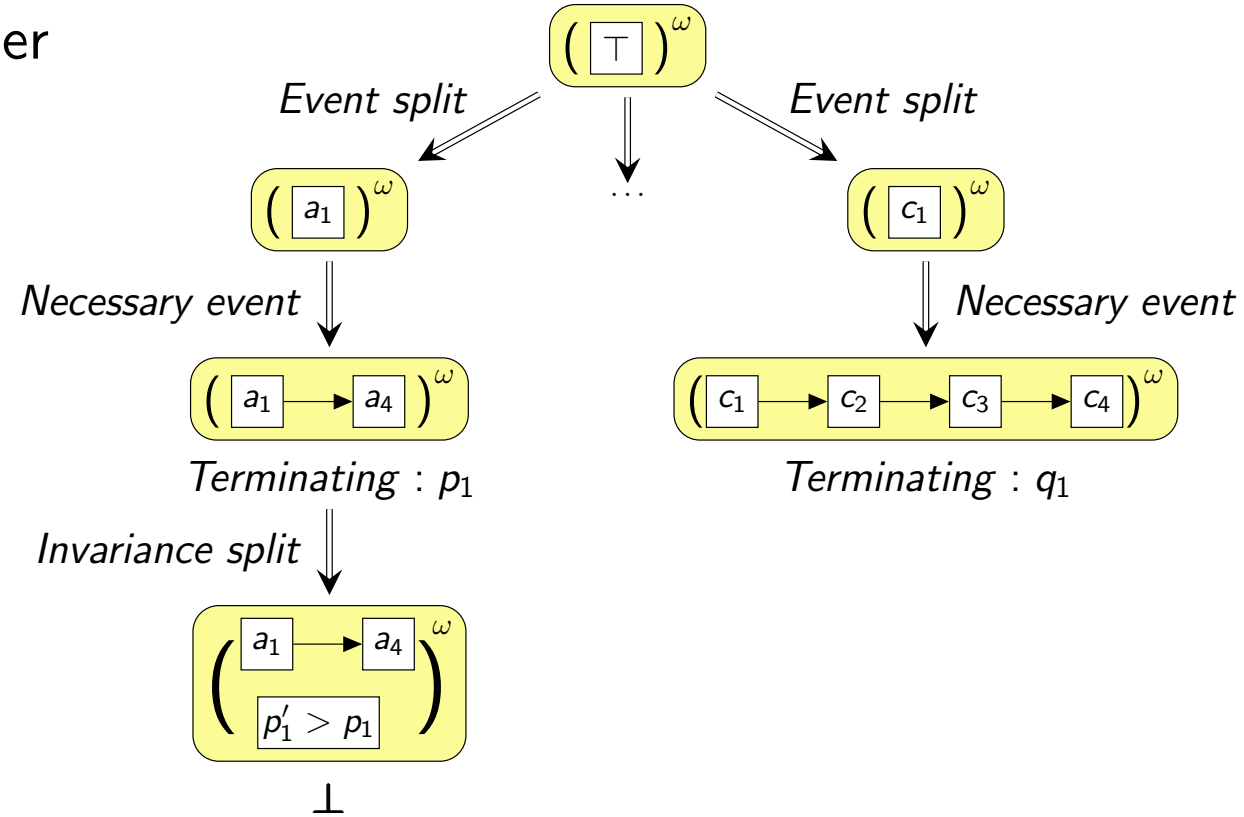
Example: Producer-Consumer



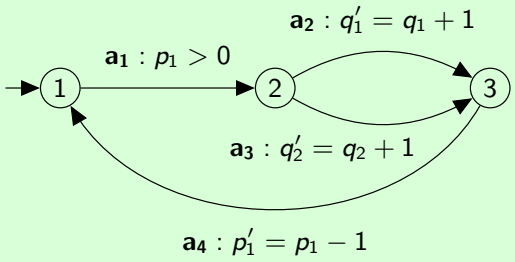


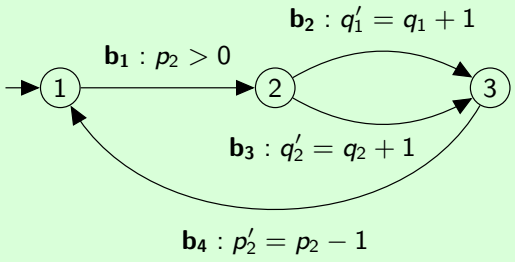


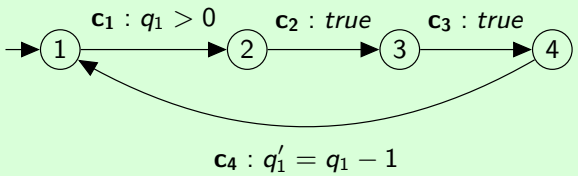


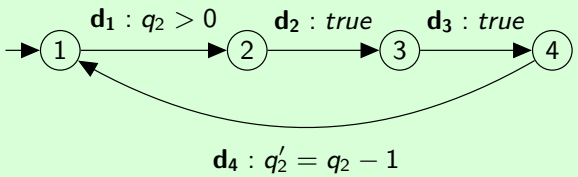


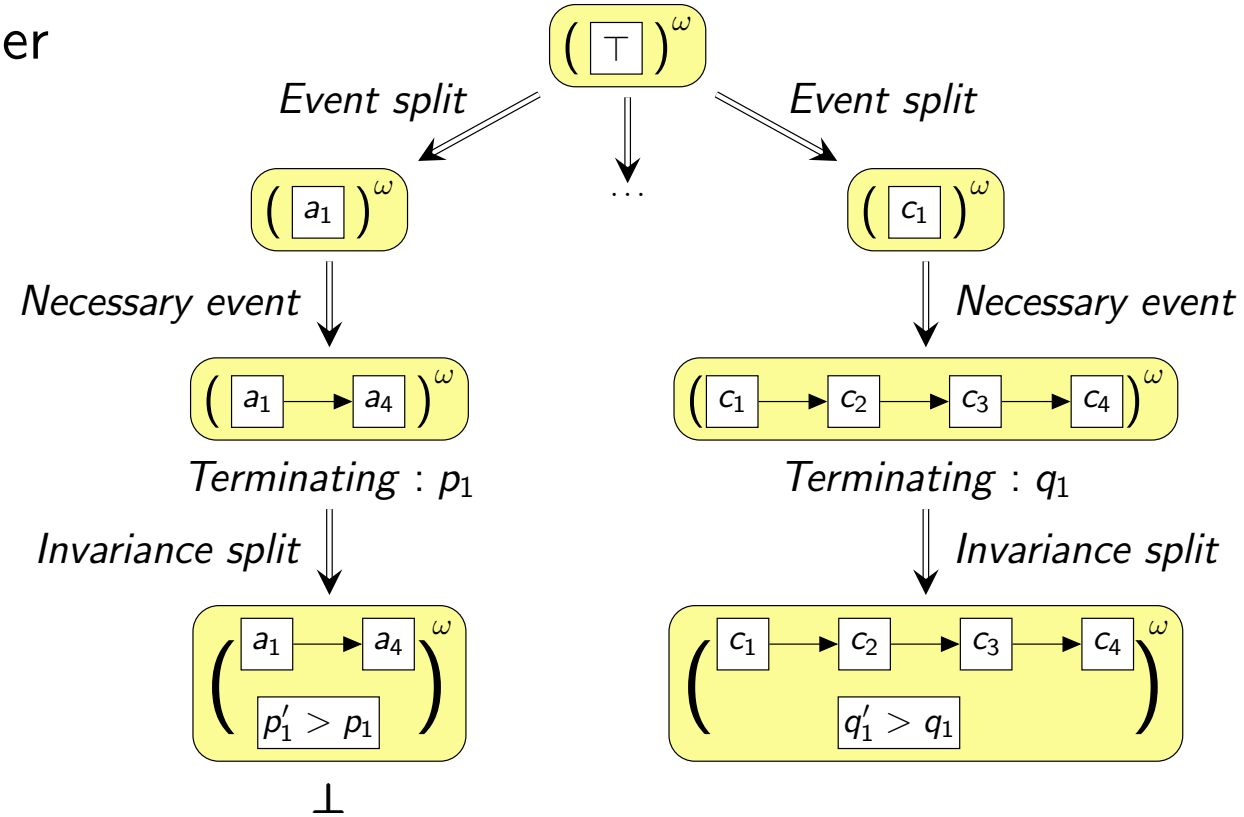
Example: Producer-Consumer



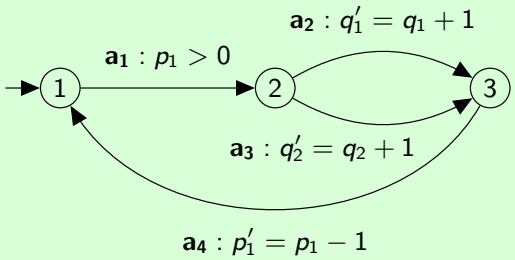


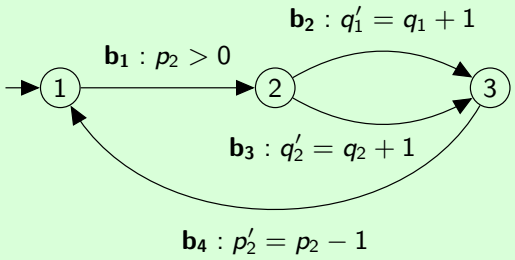


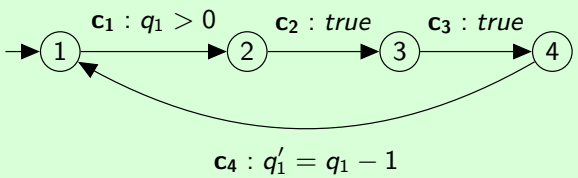


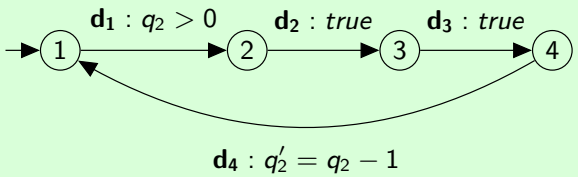


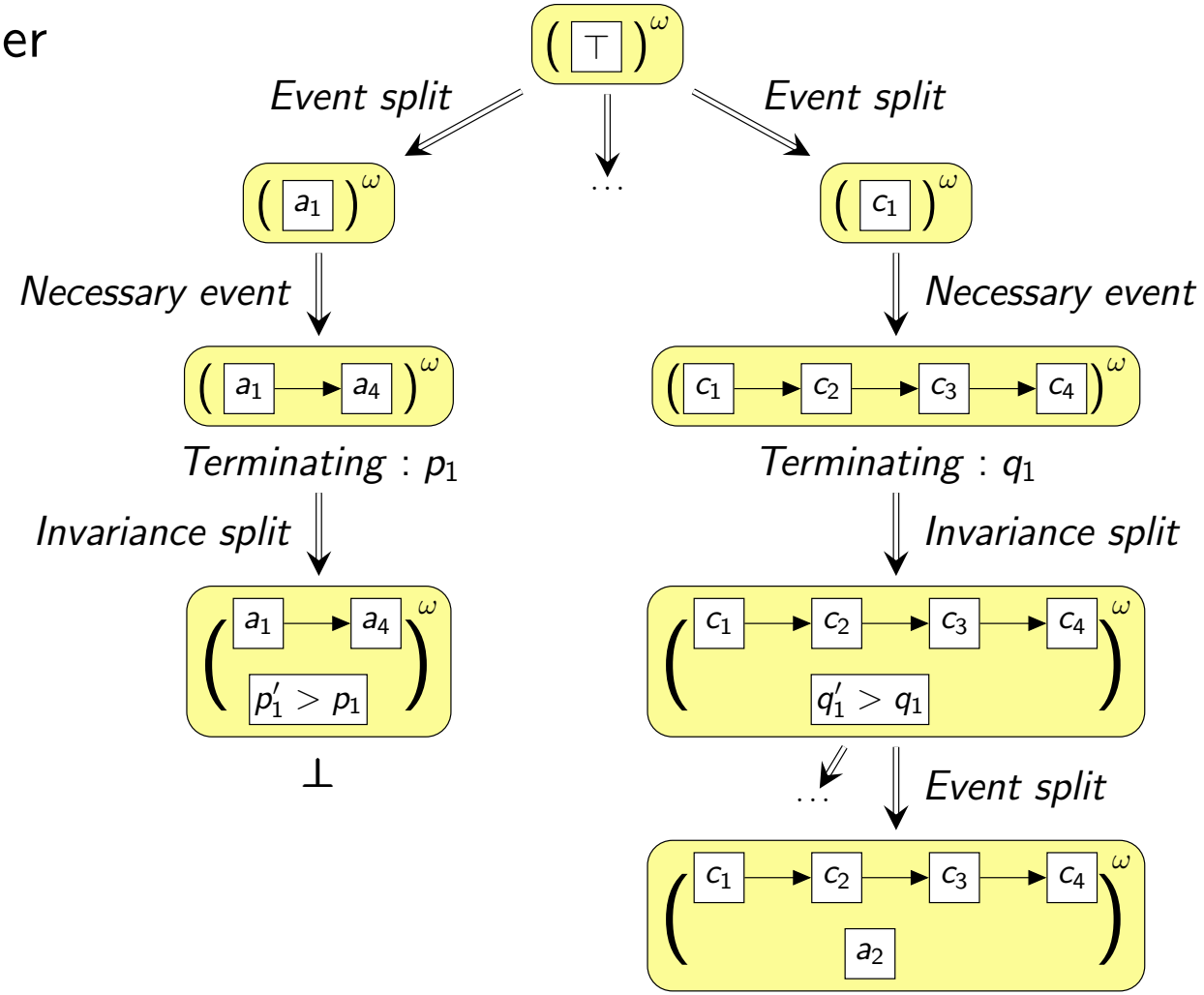
Example: Producer-Consumer



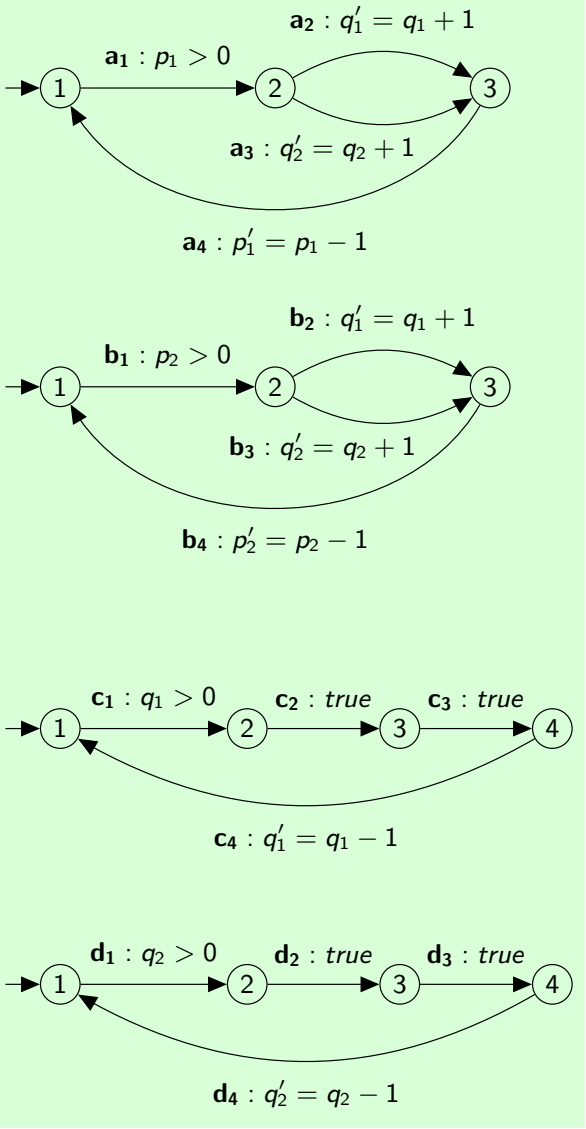


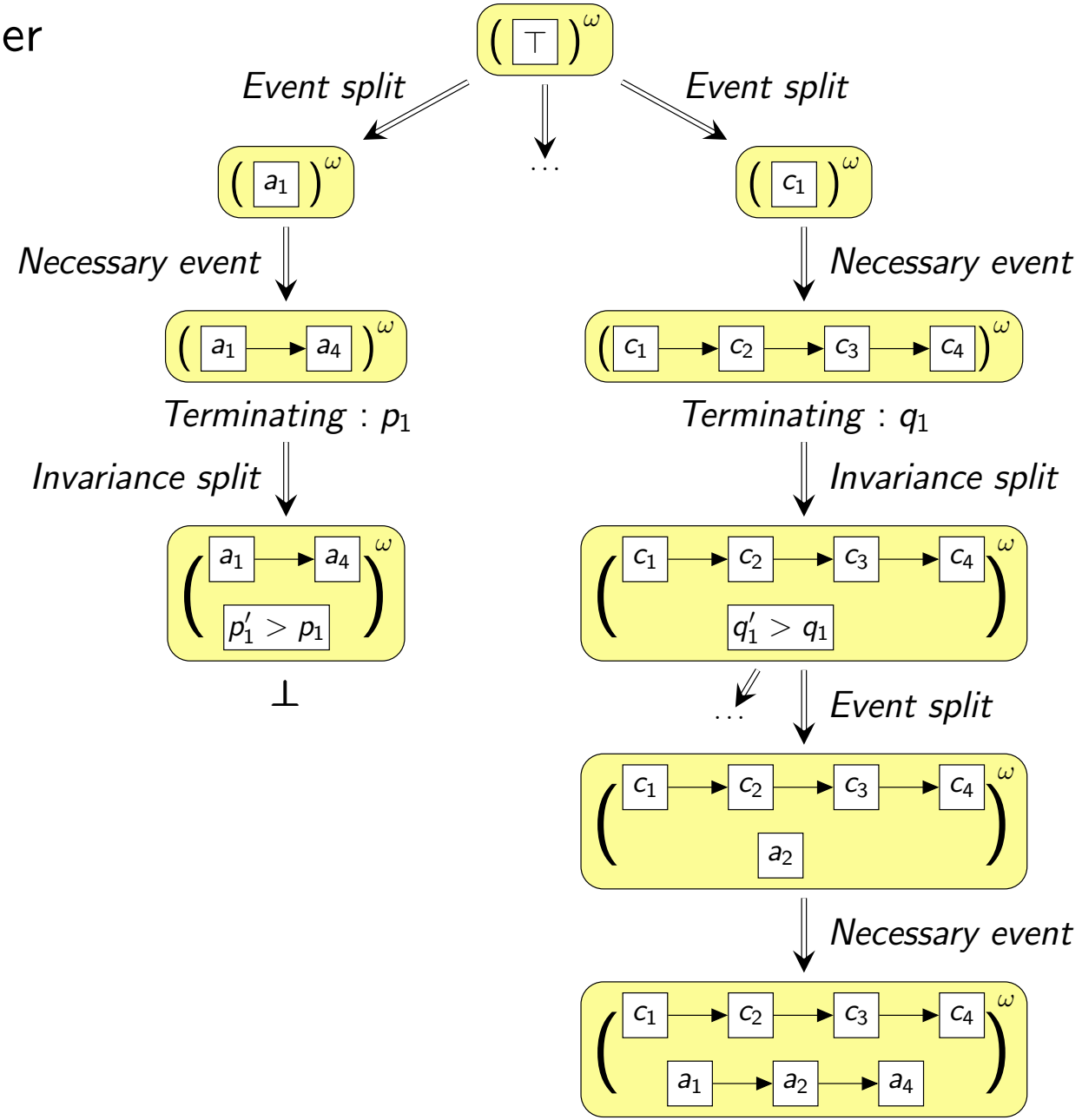




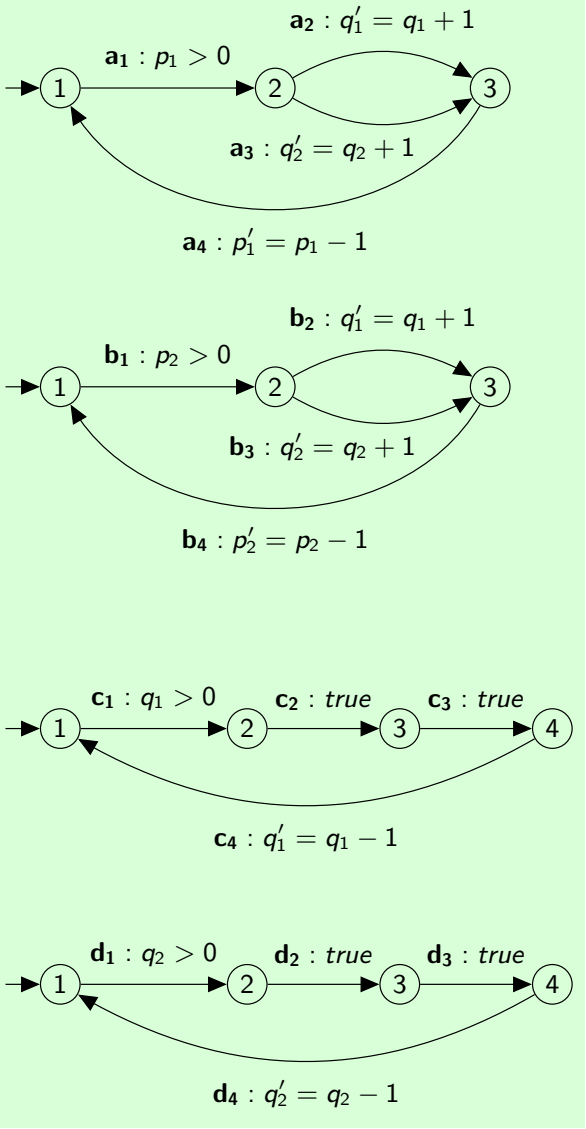


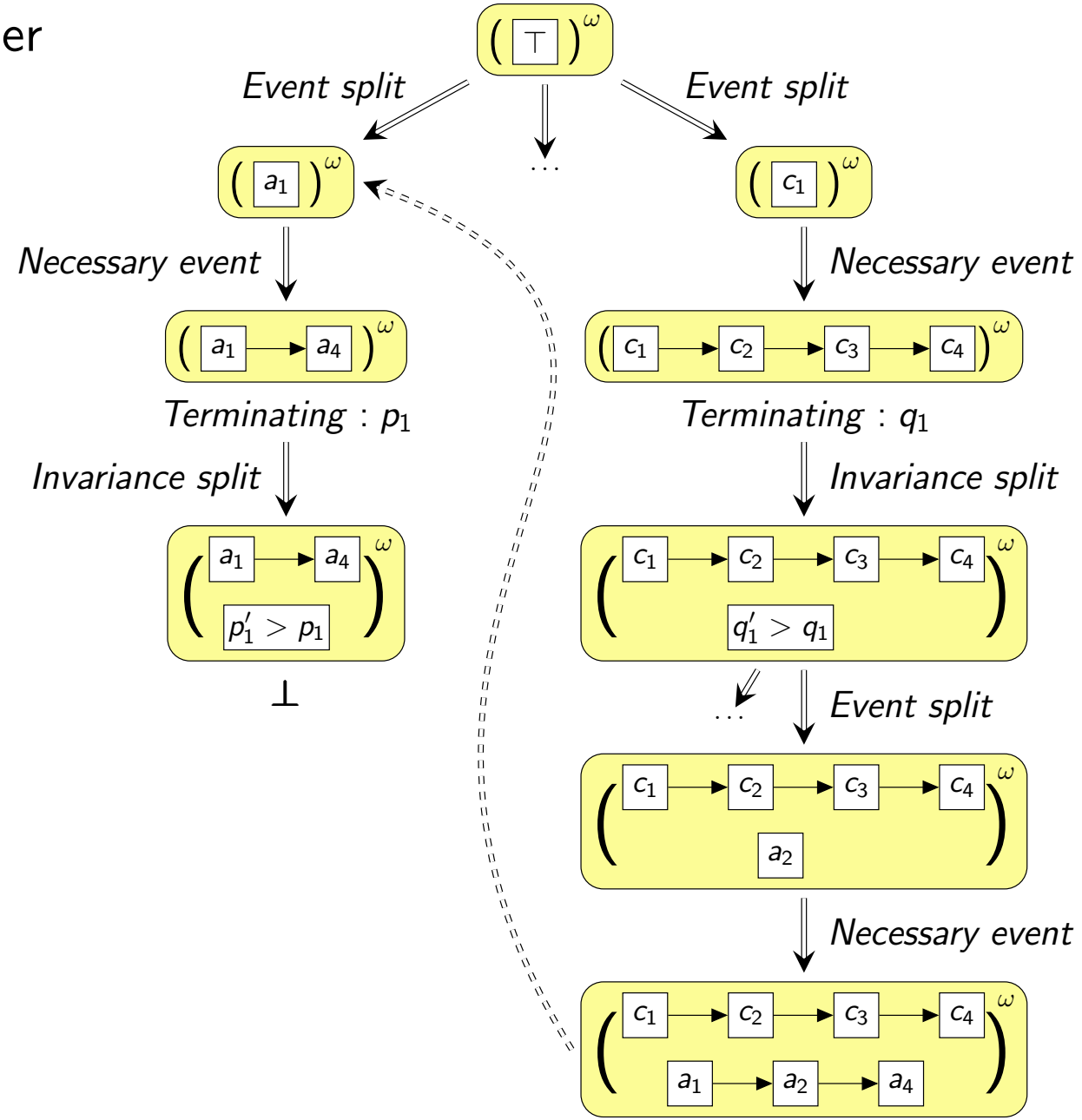
Example: Producer-Consumer





Example: Producer-Consumer



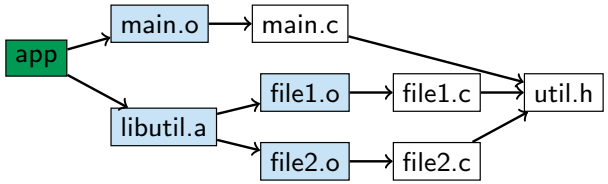


Experimental results: simple programs

	Terminator		T2		AProVE		Arctor		
Benchmark	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)	Time(s)	Mem.(MB)	Vertices
Chain 2	0.65	20	0.52	20	1.58	131	0.002	2.0	3
Chain 4	1.45	25	0.54	22	2.13	153	0.002	2.2	7
Chain 6	24.4	57	0.58	24	2.58	171	0.002	2.5	11
Chain 8	×	MO	0.63	26	3.48	210	0.002	2.5	15
Chain 20	×	MO	2.36	55	16.5	941	0.007	2.5	39
Chain 40	×	MO	40.5	288	536	1237	0.023	2.8	79
Chain 60	×	MO	Z3-TO	×	×	MO	0.063	3.0	119
Chain 80	×	MO	Z3-TO	×	×	MO	0.145	3.3	159
Chain 100	×	MO	Z3-TO	×	×	MO	0.320	3.9	199
Phase 1	×	MO	U(4.53)	48	1.60	132	0.002	2.4	2
Phase 2	×	MO	U(4.53)	48	2.16	144	0.002	2.4	11
Phase 3	×	MO	U(30.6)	301	3.83	199	0.002	2.5	20
Phase 4	×	MO	×	MO	8.89	336	0.003	2.6	29
Phase 8	×	MO	×	MO	47.0	1506	0.003	2.6	65
Phase 10	×	MO	×	MO	×	MO	0.012	2.7	83
Phase 20	×	MO	×	MO	×	MO	0.061	3.3	173
Phase 40	×	MO	×	MO	×	MO	0.35	4.0	353
Phase 60	×	MO	×	MO	×	MO	1.18	4.2	533
Phase 80	×	MO	×	MO	×	MO	3.21	5.1	713
Phase 100	×	MO	×	MO	×	MO	7.38	6.1	893
Semaphore 1	3.05	26	2.81	46	3.22	230	0.002	2.6	8
Semaphore 2	622	691	U(20.7)	219	U(6.52)	465	0.002	2.6	16
Semaphore 3	×	MO	U(15.8)	239	U(10.42)	1138	0.003	2.6	24
Semaphore 10	×	MO	U(83.5)	470	U(246)	1287	0.023	2.8	80
Semaphore 20	×	MO	×	MO	×	MO	0.073	3.3	160
Semaphore 40	×	MO	×	MO	×	MO	0.264	4.0	320
Semaphore 60	×	MO	×	MO	×	MO	0.58	4.0	480
Semaphore 80	×	MO	×	MO	×	MO	1.02	4.6	640
Semaphore 100	×	MO	×	MO	×	MO	1.59	5.1	800

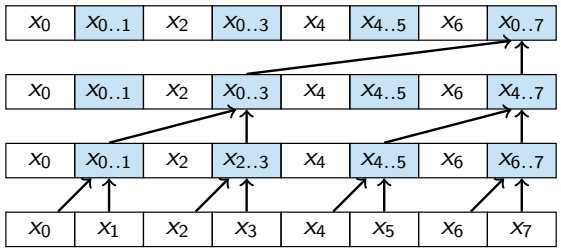
Experimental results: models of industrial programs

- Parallel compilation (GNU Make)



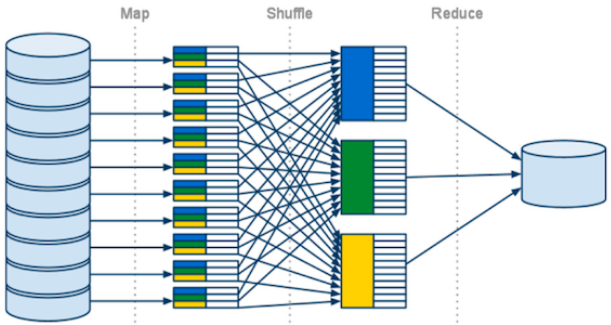
Threads	Time(s)	Mem.(MB)	Vertices
2	0.04	3.6	126
3	0.10	4.3	189
4	0.17	4.5	252
5	0.26	4.5	315
6	0.36	4.5	378
7	0.48	4.5	441
8	0.62	4.6	504
9	0.79	5.5	567
10	0.97	5.5	630

- Parallel computations in GPUs (CUDA)



Threads	Time(s)	Mem.(MB)	Vertices
2	0.04	3.3	86
3	0.09	3.7	129
4	0.15	4.3	172
5	0.24	4.5	215
6	0.33	4.5	258
7	0.45	4.6	301
8	0.58	5.5	344
9	0.72	5.5	387
10	0.88	5.5	430

- Distributed processing (Map-Reduce)



Threads	Time(s)	Mem.(MB)	Vertices
2	0.42	4.5	238
3	2.50	4.5	393
4	8.22	5.5	547
5	31.3	6.5	767
6	78.7	6.5	986
7	219	7.3	1271
8	457	8.3	1555
9	1053	9.3	1905
10	1924	11.4	2254

No other termination prover can handle even 2 threads!

Arctor

Conclusion

Our approach to causality

- dependence / independence $\Longrightarrow$ concurrent traces
- causal inferences $\Longrightarrow$ language-preserving trace transformations
- similarity recognition $\Longrightarrow$ tableau-based trace search

Conclusion

Our approach to causality

- dependence / independence $\Longrightarrow$ concurrent traces
- causal inferences $\Longrightarrow$ language-preserving trace transformations
- similarity recognition $\Longrightarrow$ tableau-based trace search

The first termination prover that scales to a large number of threads:

- **Arctor** : **A**bstraction **R**efinement of **C**oncurrent **T**emporal **Or**derings
- react.uni-saarland.de/tools/arctor/

Conclusion

Our approach to causality

- dependence / independence $\Longrightarrow$ concurrent traces
- causal inferences $\Longrightarrow$ language-preserving trace transformations
- similarity recognition $\Longrightarrow$ tableau-based trace search

The first termination prover that scales to a large number of threads:

- **Arctor** : **A**bstraction **R**efinement of **C**oncurrent **T**emporal **Or**derings
- react.uni-saarland.de/tools/arctor/

Thank you for your attention!