EMBEDDED SYSTEMS WEEK

Embedded Systems Week
www.esweek.org

Automatically transforming and relating Uppaal models of embedded systems

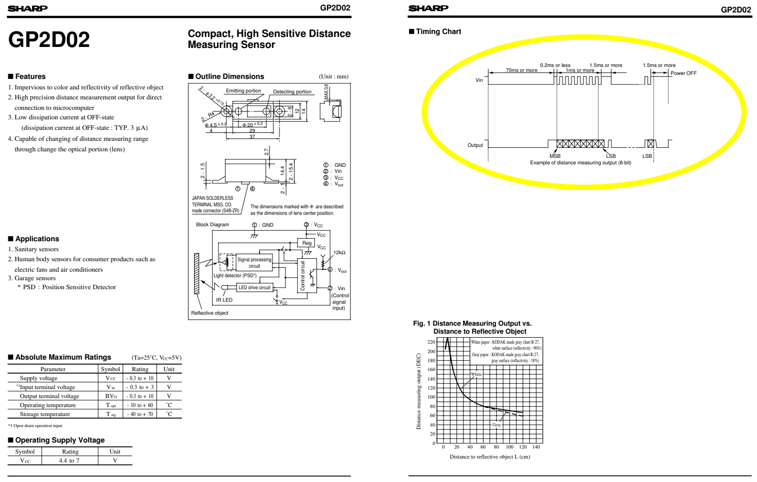
Timothy Bourke CSE UNSW and NICTA

Arcot Sowmya CSE UNSW

20081009-1739

NICTA

UNSW
THE UNIVERSITY OF NEW SOUTH WALES
SYDNEY • AUSTRALIA



SHARP GP2D02

GP2D02

Compact, High Sensitive Distance Measuring Sensor

■ **Features**

1. Impervious to color and reflectivity of reflective object
2. High precision distance measurement output for direct connection to microcomputer
3. Low dissipation current at OFF-state (dissipation current at OFF-state : TYP. 3 µA)
4. Capable of changing of distance measuring range through change the optical portion (lens)

■ **Outline Dimensions** (Unit : mm)

■ **Applications**

1. Sanitary sensors
2. Human body sensors for consumer products such as electric fans and air conditioners
3. Garage sensors

* PSD : Position Sensitive Detector

■ **Absolute Maximum Ratings** ($Ta=25°C$, $V_{CC}=5V$)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	- 0.3 to + 10	V
*1 Input terminal voltage	V_{in}	- 0.3 to + 3	V
Output terminal voltage	BV_O	- 0.3 to + 10	V
Operating temperature	T_{opr}	- 10 to + 60	°C
Storage temperature	T_{stg}	- 40 to + 70	°C

*1 Open drain operation input

■ **Operating Supply Voltage**

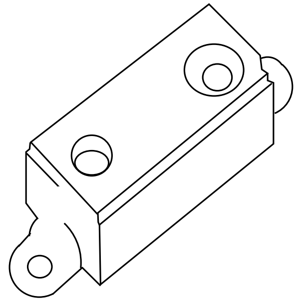
Symbol	Rating	Unit
V_{CC}	4.4 to 7	V

SHARP GP2D02

■ **Timing Chart**

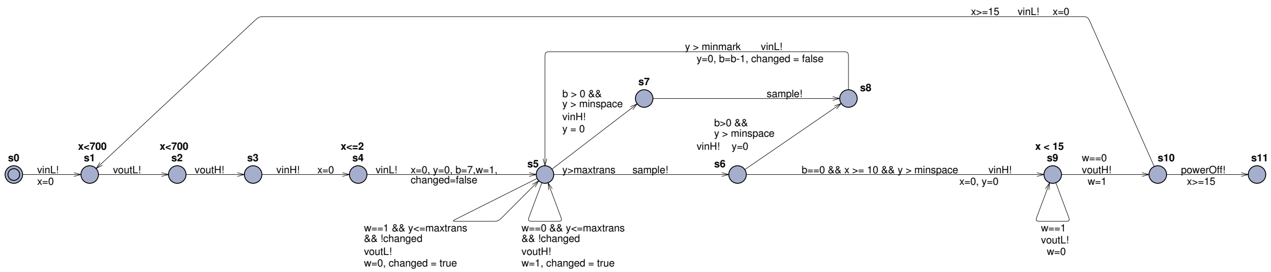
Fig. 1 Distance Measuring Output vs. Distance to Reflective Object

TIMEDIAG

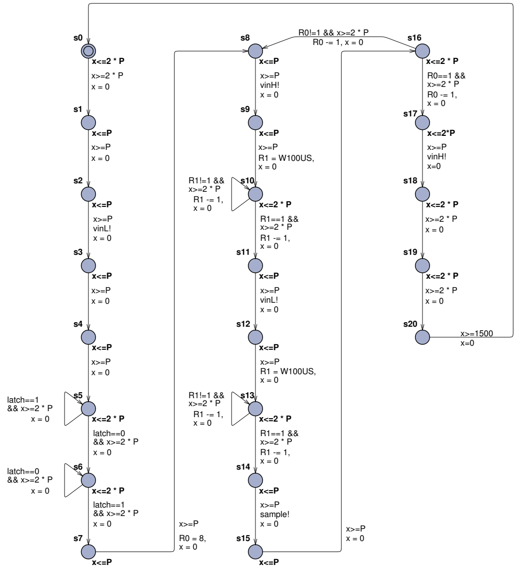


MCS51

```
VIN EQU P1.0            LOOP:  SETB VIN
VOUT EQU P1.1                  MOV R1, #W100US
W100US EQU 50                  DJNZ R1, *
                               CLR VIN
RREAD:  PUSH IE                MOV R1, #W100US
        CLR EA                 DJNZ R1, *
        CLR VIN                MOV VOUT, C
        NOP                    RLC A
        NOP                    DJNZ R0, LOOP
        JB VOUT, *             SETB VIN
        JNB VOUT, *            POP IE
        MOV R0, #8             RET
```

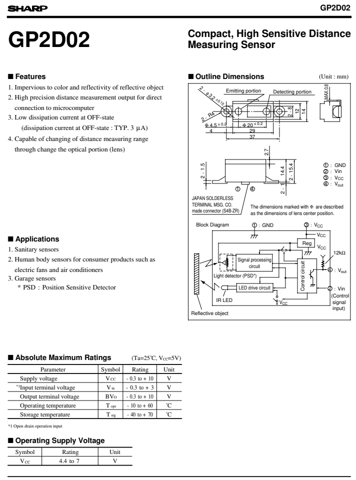


TIMEDIAG



MCS51

```
VIN EQU P1.0                LOOP:  SETB VIN
VOUT EQU P1.1                      MOV R1, #W100US
W100US EQU 50                      DJNZ R1, *
                                   CLR VIN
RREAD:  PUSH IE                    MOV R1, #W100US
        CLR EA                     DJNZ R1, *
        CLR VIN                    MOV VOUT, C
        NOP                        RLC A
        NOP                        DJNZ R0, LOOP
        JB VOUT, *                 SETB VIN
        JNB VOUT, *                POP IE
        MOV R0, #8                 RET
```


SHARP GP2D02

GP2D02

Compact, High Sensitive Distance Measuring Sensor

■ Features

1. Impervious to color and reflectivity of reflective object
2. High precision distance measurement output for direct connection to microcomputer
3. Low dissipation current at OFF-state (dissipation current at OFF-state : TYP. 3 µA)
4. Capable of changing of distance measuring range through change the optical portion (lens)

■ Outline Dimensions (Unit : mm)

■ Applications

1. Sanitary sensors
2. Human body sensors for consumer products such as electric fans and air conditioners
3. Garage sensors

* PSD : Position Sensitive Detector

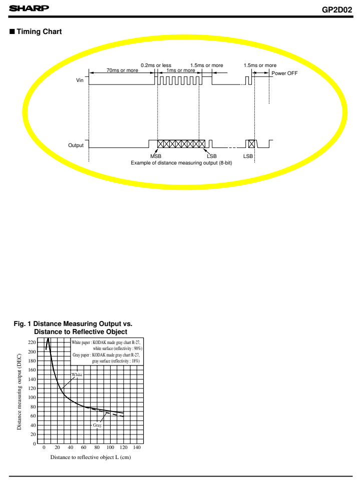
■ Absolute Maximum Ratings (Ta=25˚C, V_{CC}=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	- 0.3 to + 10	V
*1 Input terminal voltage	V_{in}	- 0.3 to + 3	V
Output terminal voltage	BV_O	- 0.3 to + 10	V
Operating temperature	T_{opr}	- 10 to + 60	˚C
Storage temperature	T_{stg}	- 40 to + 70	˚C

*1 Open drain operation input

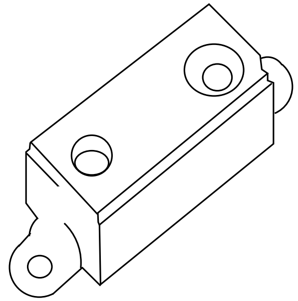
■ Operating Supply Voltage

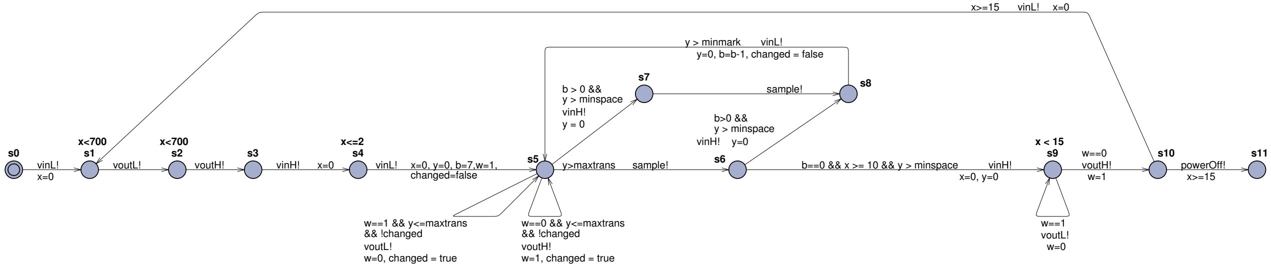
Symbol	Rating	Unit
V_{CC}	4.4 to 7	V

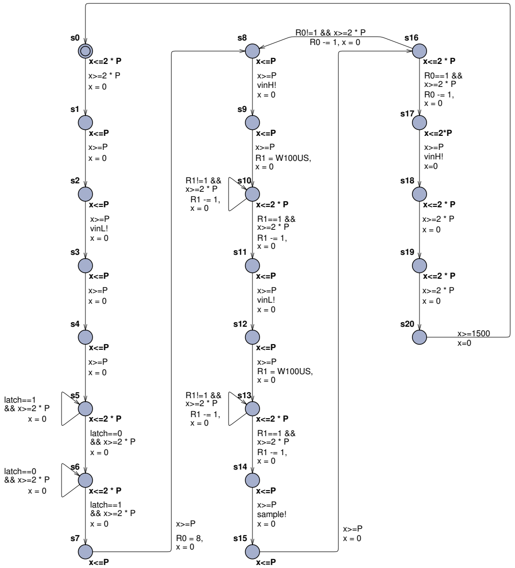

SHARP GP2D02

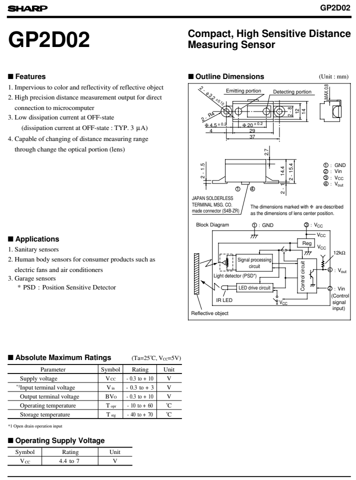
■ Timing Chart

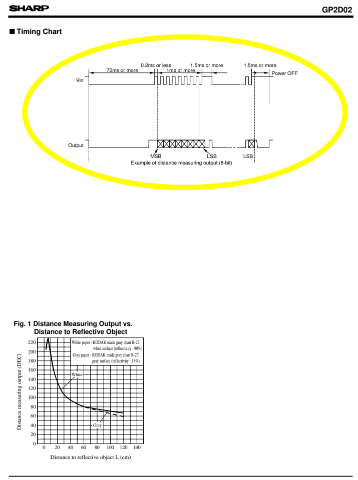
Fig. 1 Distance Measuring Output vs. Distance to Reflective Object











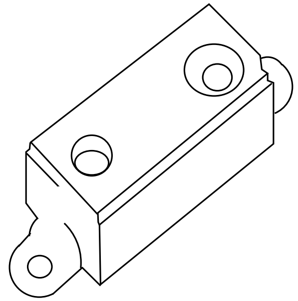
TIMEDIAG

VI

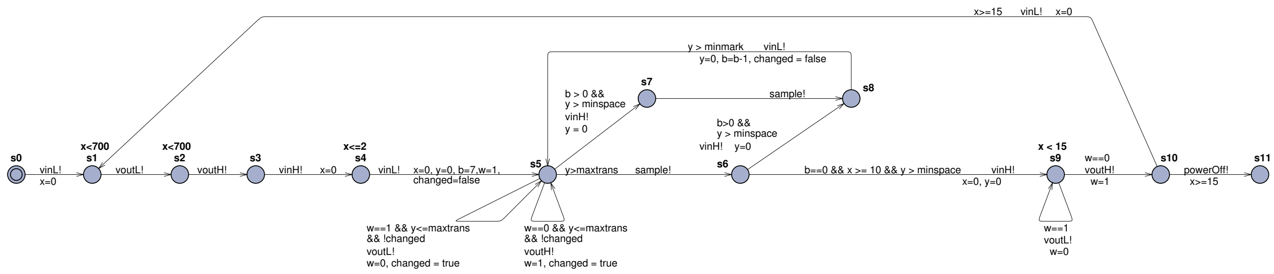
DRIVER || SENSOR

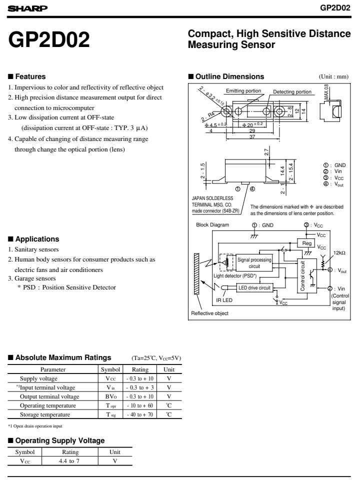
VI

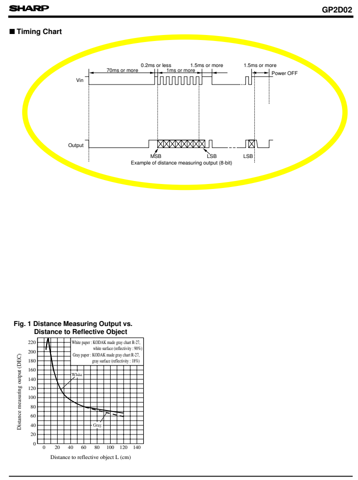
MCS51



```
VIN EQU P1.0            LOOP:  SETB VIN
VOUT EQU P1.1                  MOV R1, #W100US
W100US EQU 50                  DJNZ R1, *
                               CLR VIN
RREAD:  PUSH IE                MOV R1, #W100US
        CLR EA                 DJNZ R1, *
        CLR VIN                MOV VOUT, C
        NOP                    RLC A
        NOP                    DJNZ R0, LOOP
        JB VOUT, *             SETB VIN
        JNB VOUT, *            POP IE
        MOV R0, #8             RET
```

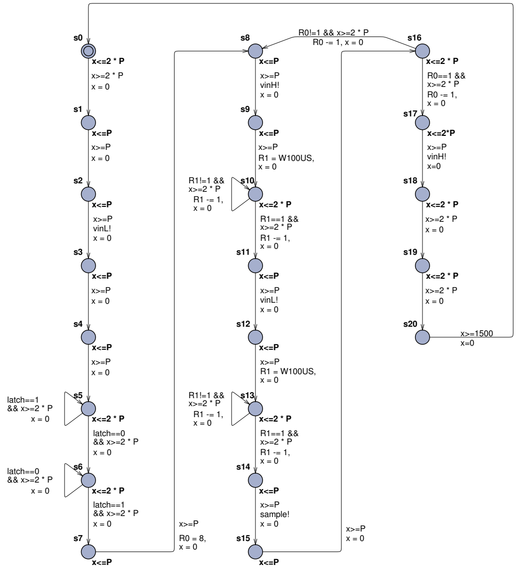




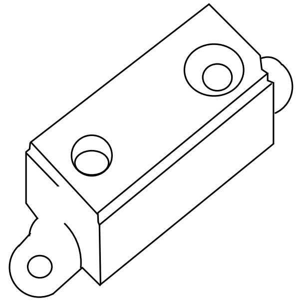


TIMEDIAG

$\leq$



DRIVER $\parallel$ SENSOR



$\leq$

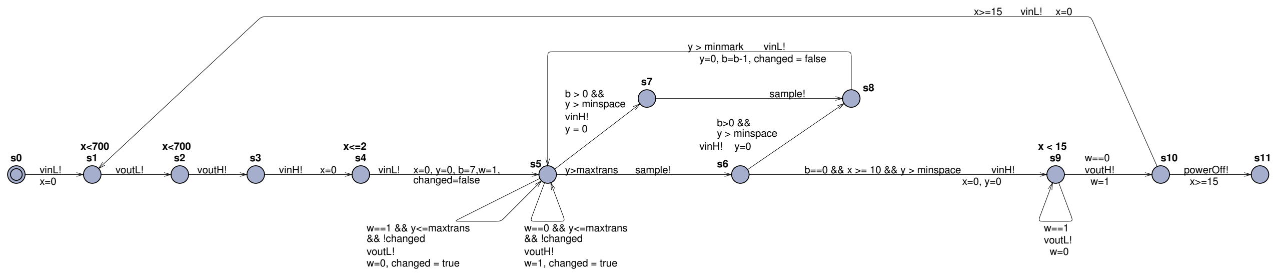
MCS51

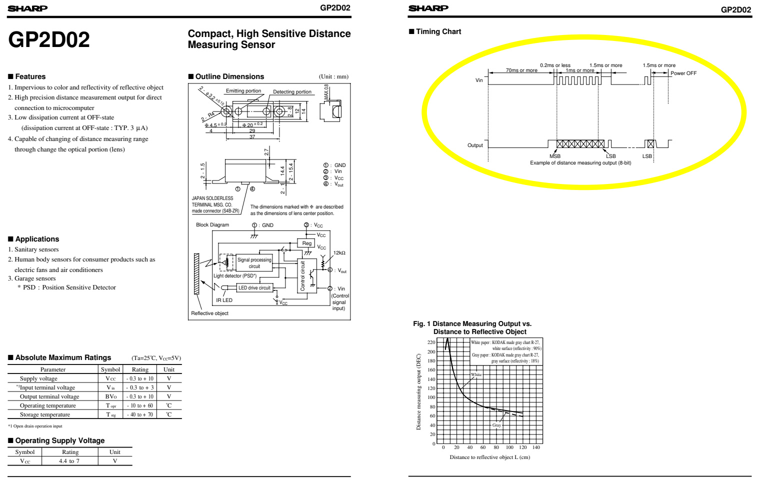
```
VIN EQU P1.0                LOOP:  SETB VIN
VOUT EQU P1.1                      MOV R1, #W100US
W100US EQU 50                      DJNZ R1, *
                                   CLR VIN
RREAD: PUSH IE                     MOV R1, #W100US
       CLR EA                      DJNZ R1, *
       CLR VIN                     MOV VOUT, C
       NOP                         RLC A
       NOP                         DJNZ R0, LOOP
       JB VOUT, *                  SETB VIN
       JNB VOUT, *                 POP IE
       MOV R0, #8                  RET
```

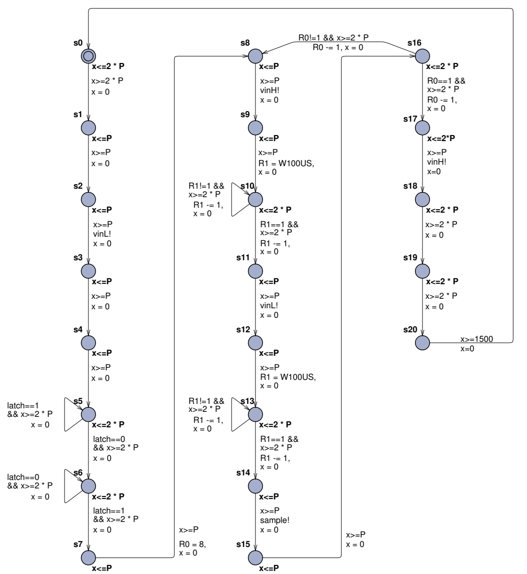
implements

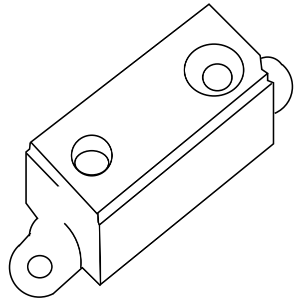
$$\mathcal{I} \leq \mathcal{S} \quad \text{if} \quad ttraces(\mathcal{I}) \subseteq ttraces(\mathcal{S})$$

abstracts









TIMEDIAG

⩾

DRIVER ‖ SENSOR

⩾

MCS51

```
VIN EQU P1.0            LOOP:  SETB VIN
VOUT EQU P1.1                  MOV R1, #W100US
W100US EQU 50                  DJNZ R1, *
                               CLR VIN
RREAD: PUSH IE                 MOV R1, #W100US
       CLR EA                  DJNZ R1, *
       CLR VIN                 MOV VOUT, C
       NOP                     RLC A
       NOP                     DJNZ R0, LOOP
       JB VOUT, *              SETB VIN
       JNB VOUT, *             POP IE
       MOV R0, #8              RET
```

implements

must be deterministic

$$\mathcal{I} \leq \mathcal{S} \quad \text{if} \quad \mathit{ttraces}(\mathcal{I}) \subseteq \mathit{ttraces}(\mathcal{S})$$

abstracts

$\Uparrow$

sometimes decidable (in Uppaal)

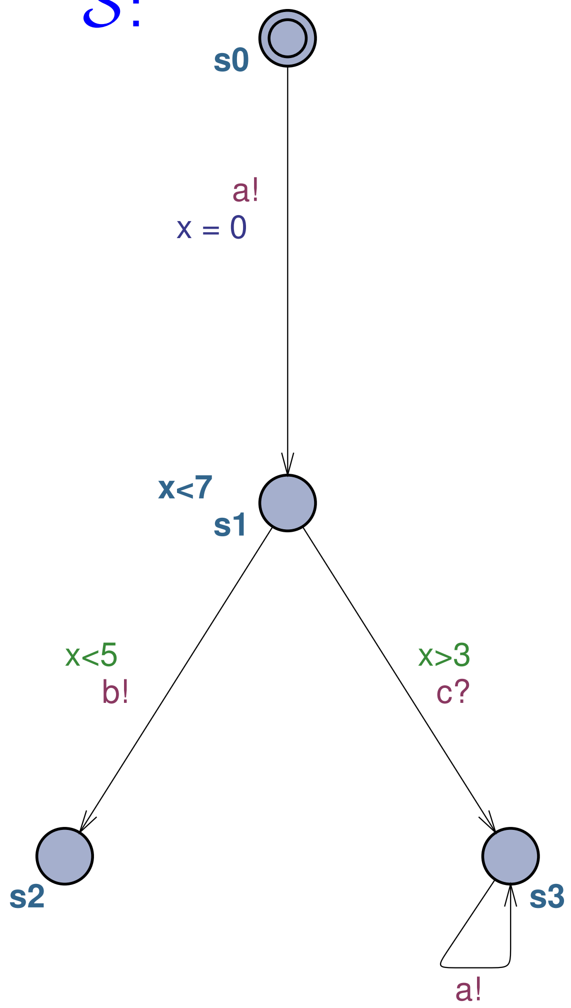
[Alur and Dill, 1994]

Presentation Outline

- ⇒ Testing timed trace inclusion
- Automation and Uppaal features
- Basic guards
- Selection bindings
- Quantifiers
- Channel arrays
- Implementation
- Summary

Testing timed trace inclusion

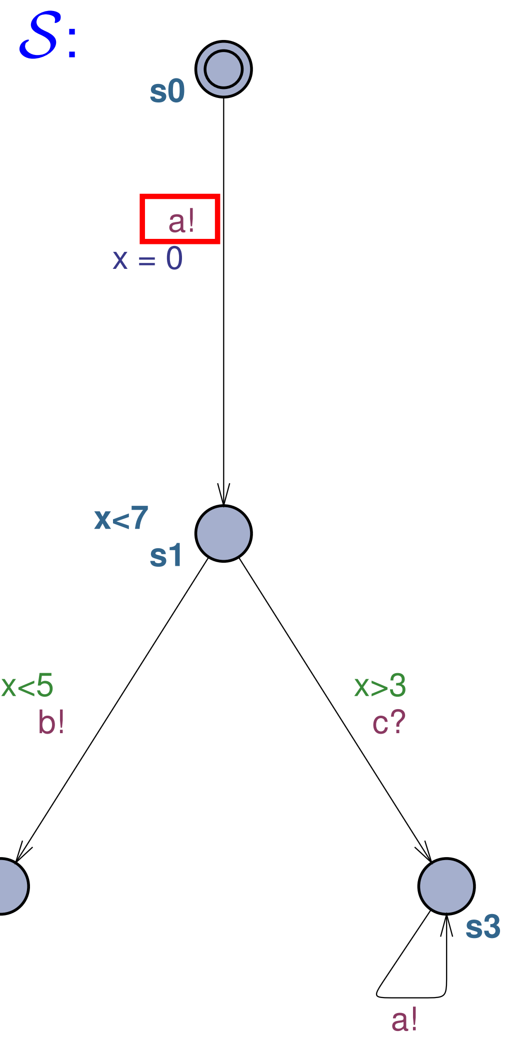
$\mathcal{S}$:



Uppaal [Larsen et al., 1997]:

- Timed (safety) Automata [Henzinger et al., 1992]
- Dense time
- Local variables
- Modelling: graphical & C-like

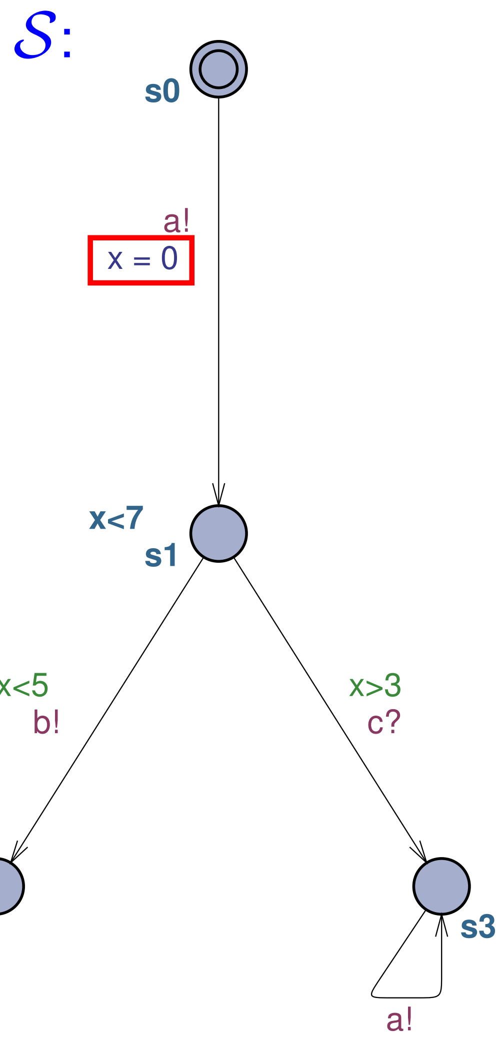
Testing timed trace inclusion



Uppaal [Larsen et al., 1997]:

- Timed (safety) Automata [Henzinger et al., 1992]
- Dense time
- Local variables
- Modelling: graphical & C-like

Testing timed trace inclusion

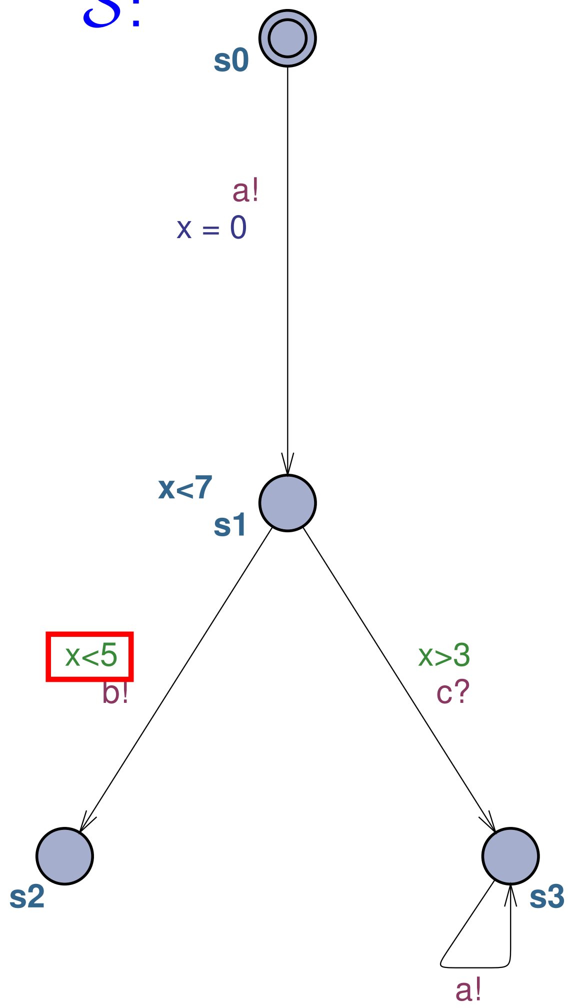


Uppaal [Larsen et al., 1997]:

- Timed (safety) Automata [Henzinger et al., 1992]
- Dense time
- Local variables
- Modelling: graphical & C-like

Testing timed trace inclusion

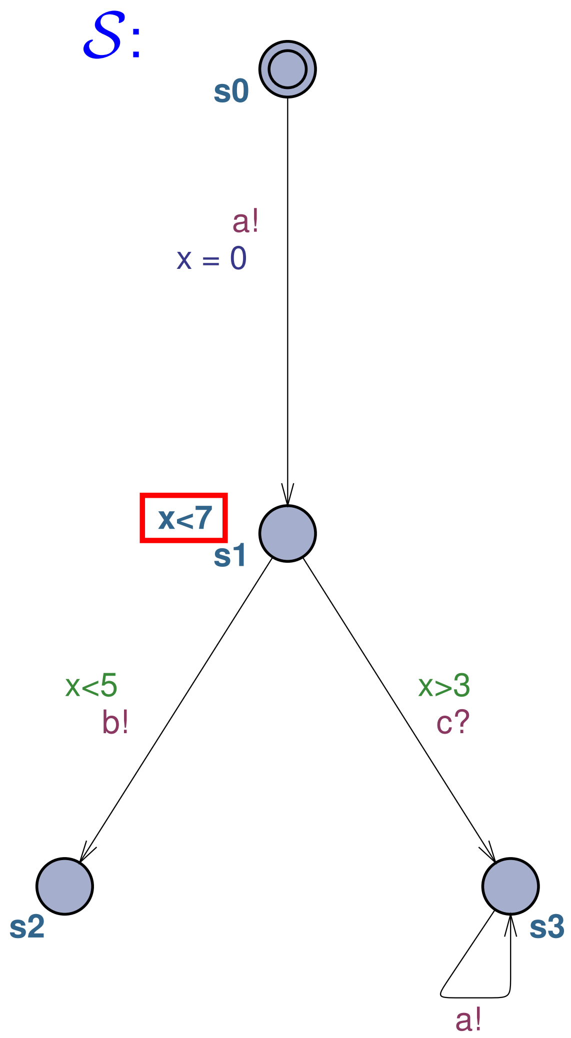
$\mathcal{S}$:



Uppaal [Larsen et al., 1997]:

- Timed (safety) Automata [Henzinger et al., 1992]
- Dense time
- Local variables
- Modelling: graphical & C-like

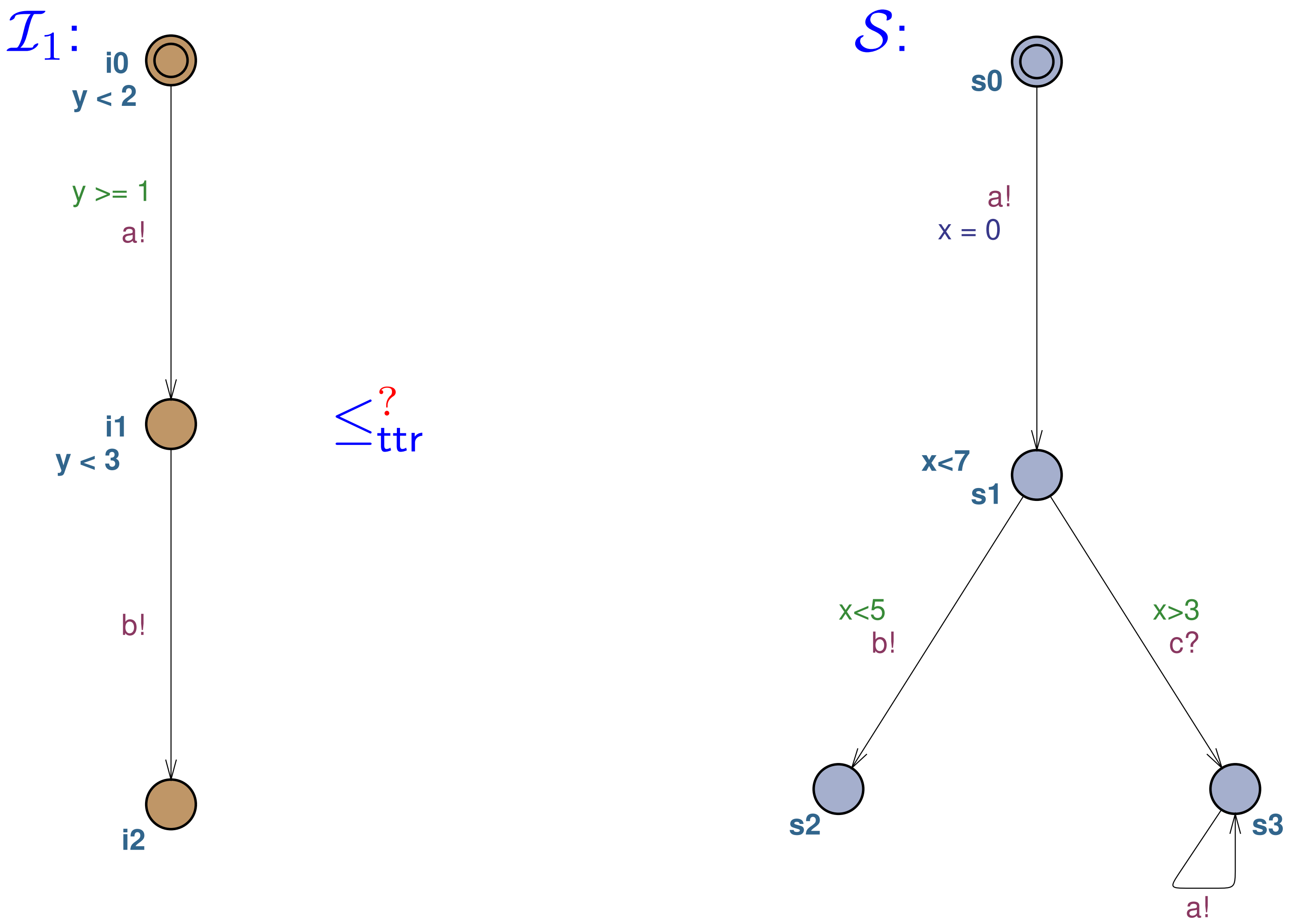
Testing timed trace inclusion



Uppaal [Larsen et al., 1997]:

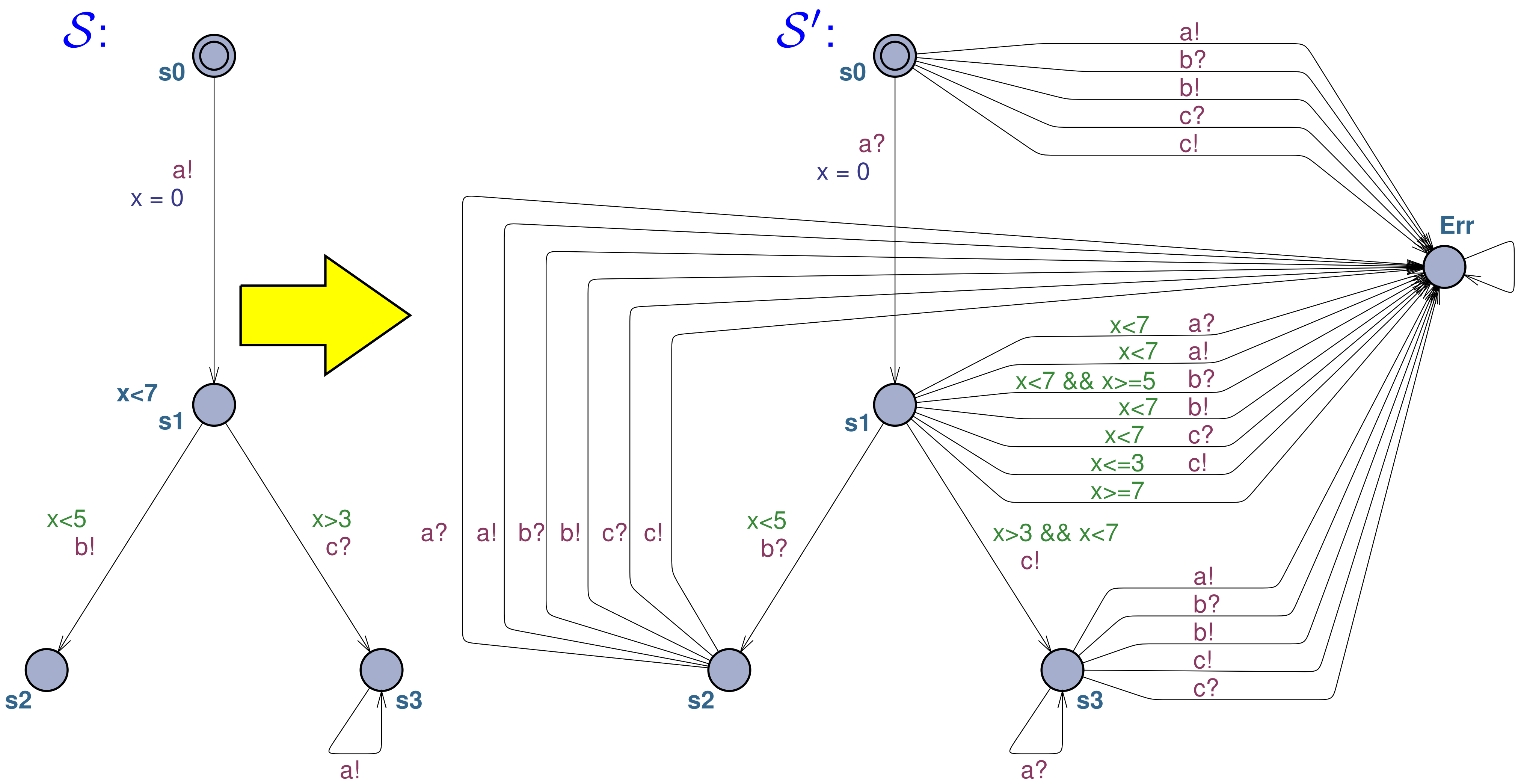
- Timed (safety) Automata [Henzinger et al., 1992]
- Dense time
- Local variables
- Modelling: graphical & C-like

Testing timed trace inclusion



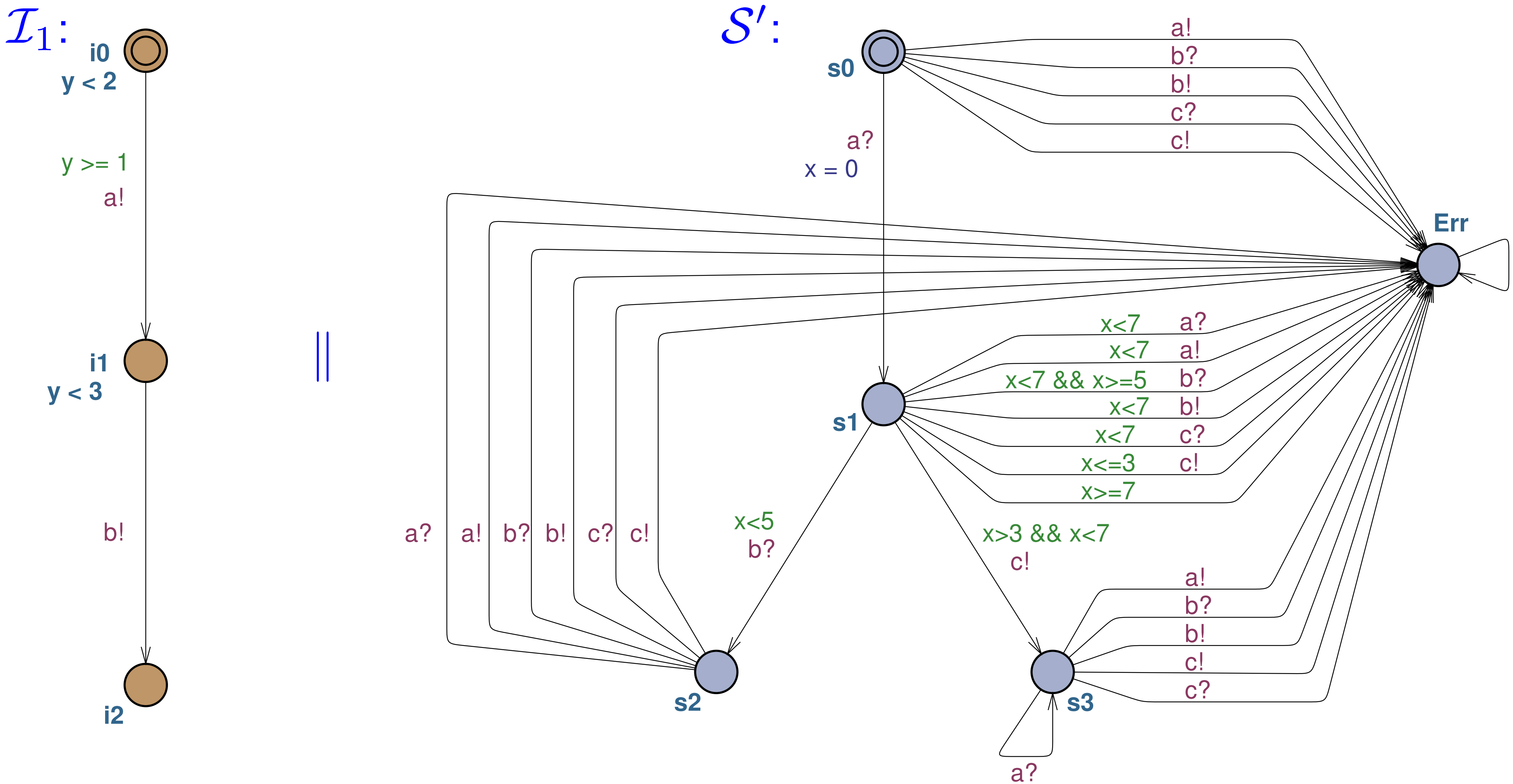
- Does $\mathcal{I}_1$ implement $\mathcal{S}$?

Testing timed trace inclusion



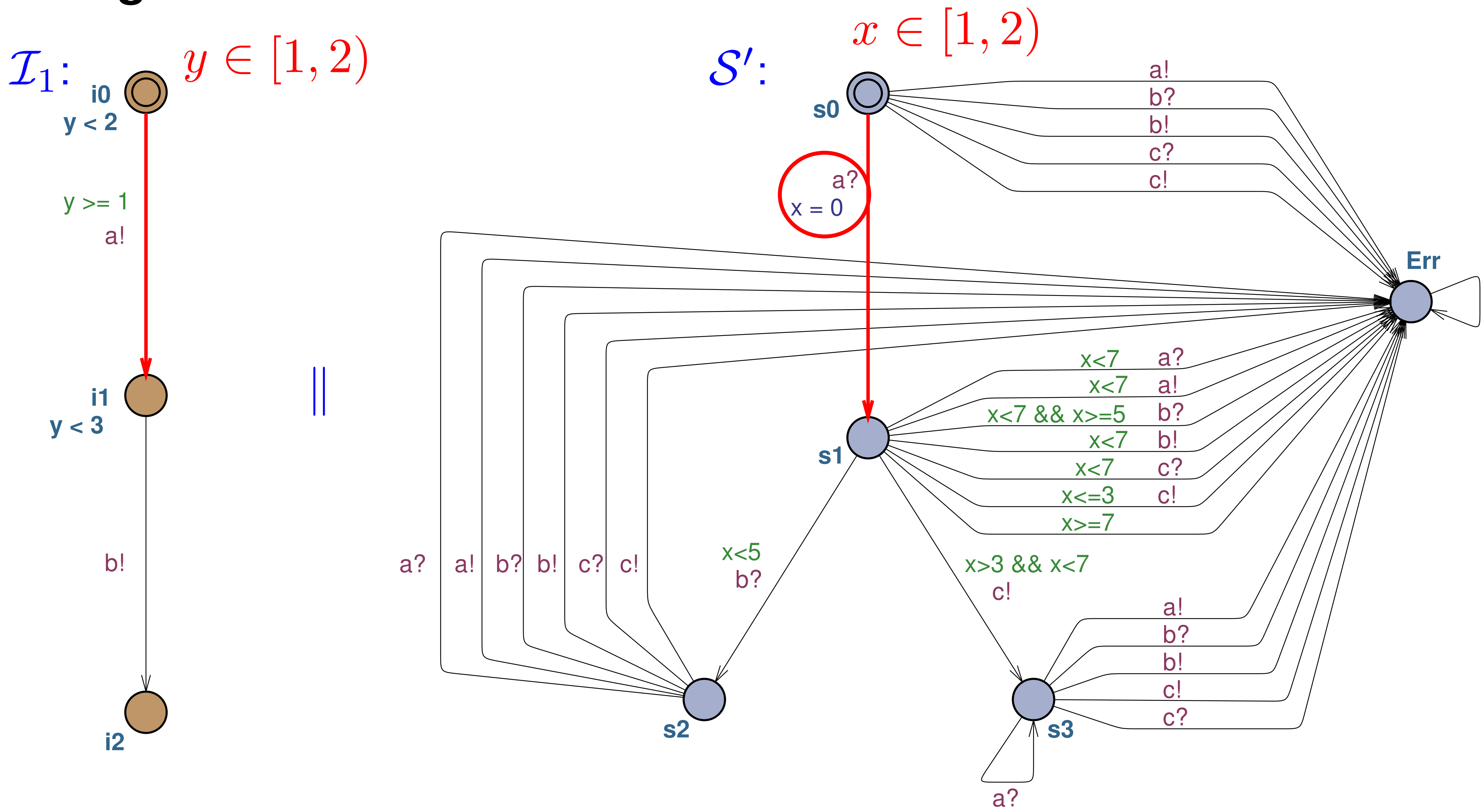
- $\mathcal{S}'$ is a testing automaton for $\mathcal{S}$
- New Err state
- Actions are complemented
- Invariants are shifted

Testing timed trace inclusion

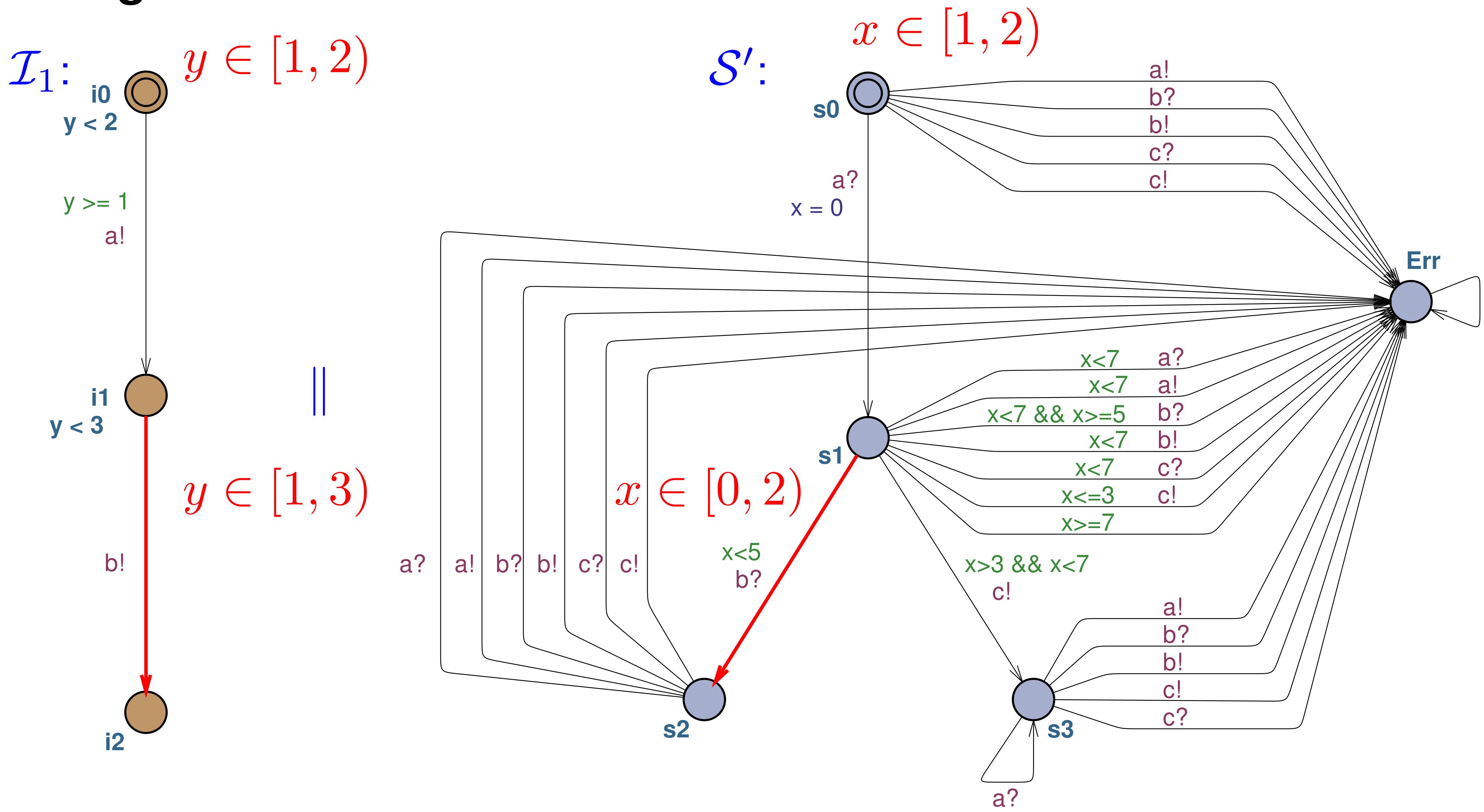


- Run $\mathcal{S}'$ in parallel with $\mathcal{I}_1$
- Is Err reachable?

Testing timed trace inclusion

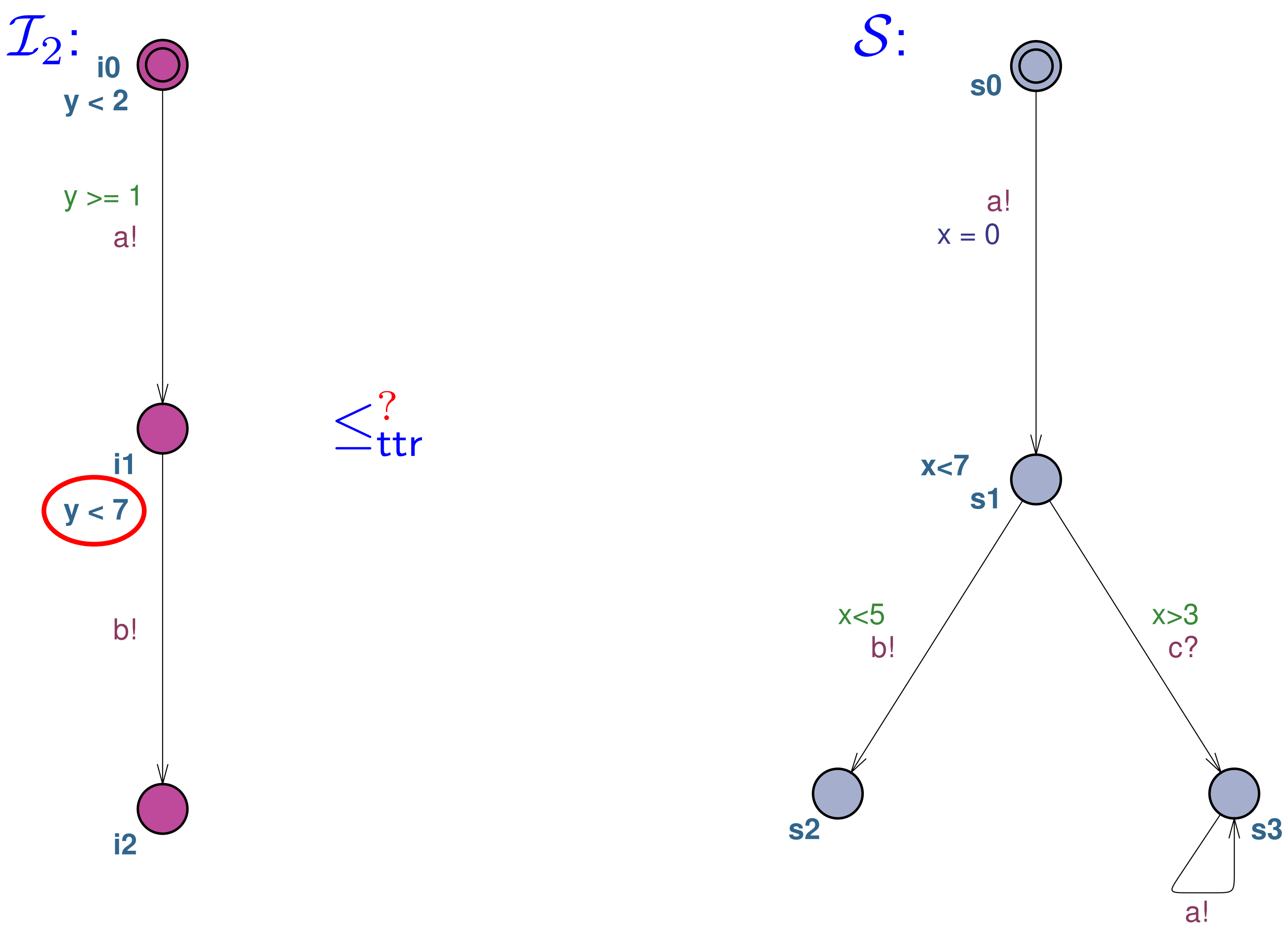


Testing timed trace inclusion



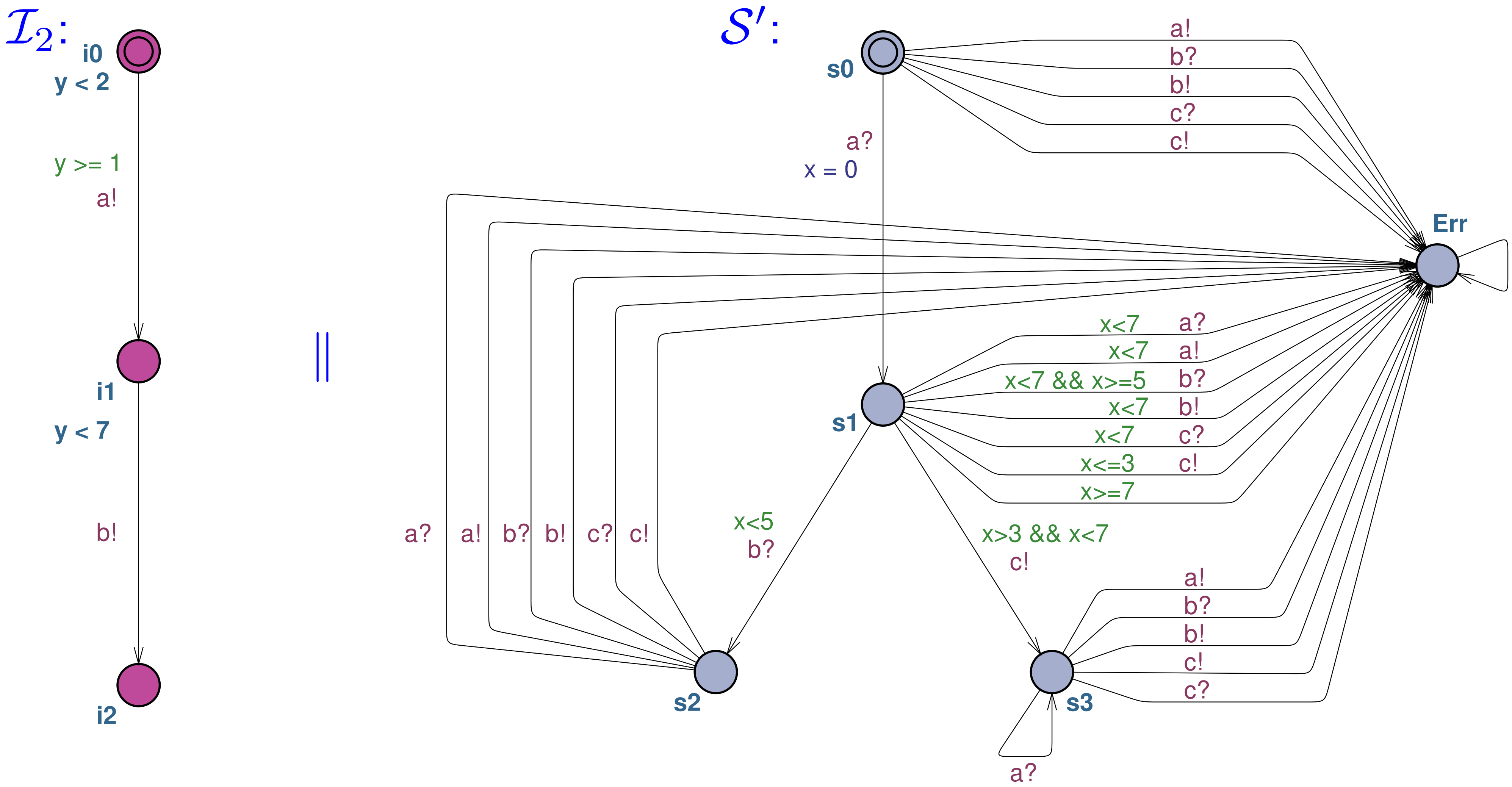
- $y - x \in [1,2)$
- Err is not reachable, therefore $\mathcal{I}_1 \leq_{\mathrm{ttr}} \mathcal{S}$

Testing timed trace inclusion



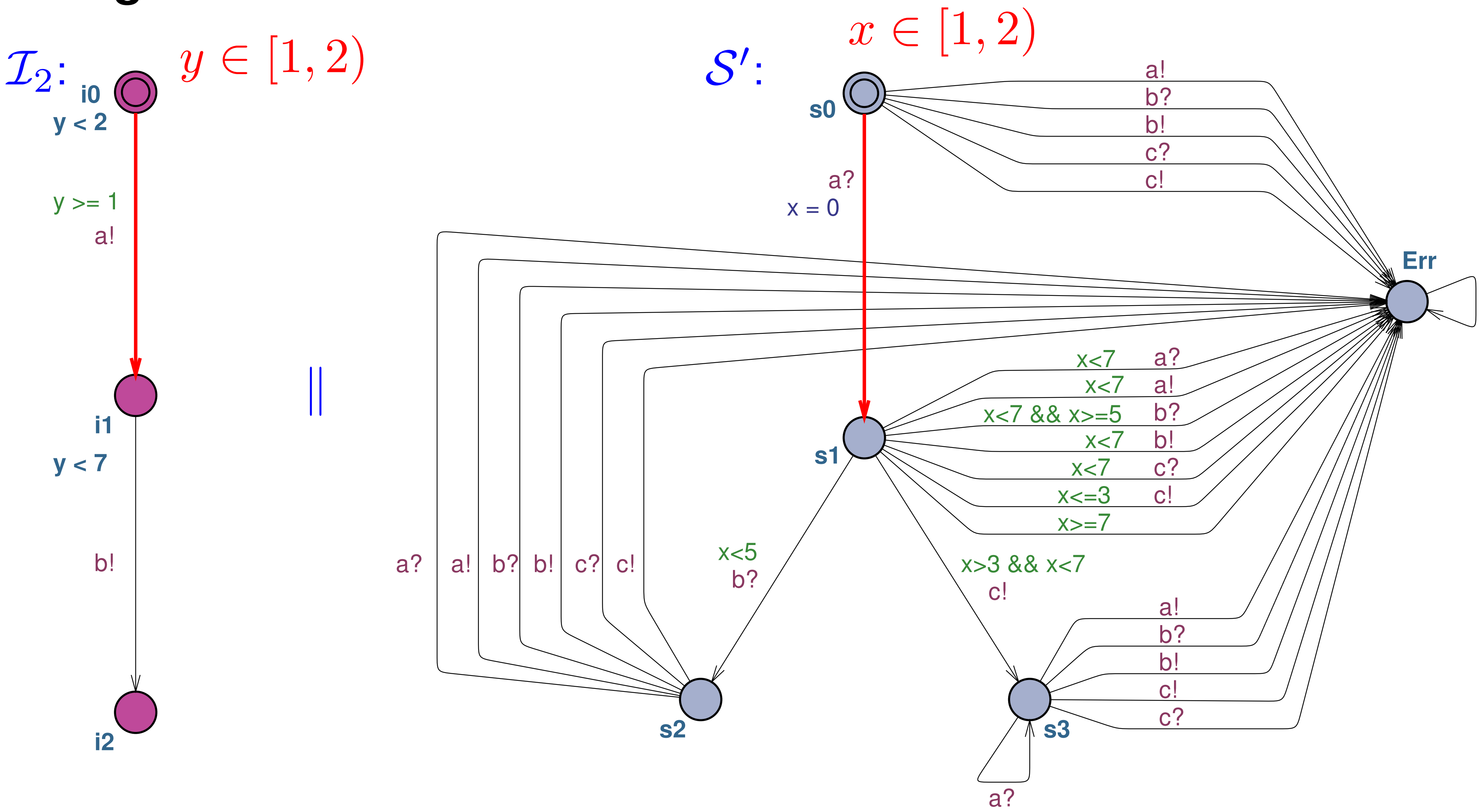
- Change the invariant on i1, from $y < 3$ to $y < 7$.
- Does $\mathcal{I}_2$ implement $\mathcal{S}$?

Testing timed trace inclusion



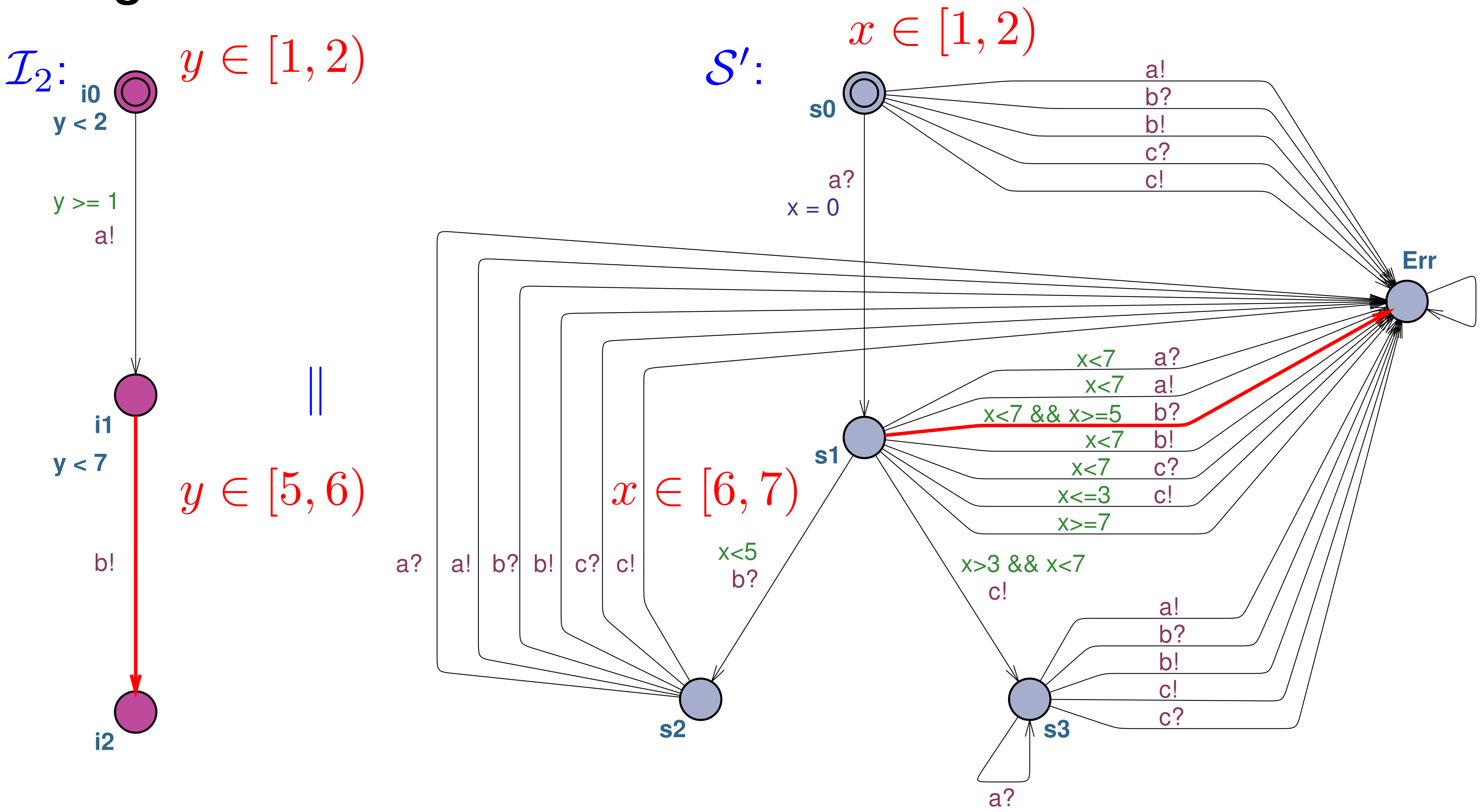
- Use the same test automaton $\mathcal{S}'$
- Is Err reachable?

Testing timed trace inclusion



- So far so good...

Testing timed trace inclusion



- $y - x \in [1, 2)$
- Err is reachable, therefore $\mathcal{I} \not\leq_{\mathsf{ttr}} \mathcal{S}$, counterexamples: $\xrightarrow{[1,2)} a! \xrightarrow{[5,6)} b!$

Automation and Uppaal features

Why automate?

- Construction is tedious and error-prone,
- Testing reveals flaws which require fixes, then more testing. . .

Automation and Uppaal features

Why automate?

- Construction is tedious and error-prone,
- Testing reveals flaws which require fixes, then more testing. . .

Automate existing construction

[Stoelinga, 2002, Jensen et al., 2000]

But what about extra Uppaal features?

urgent nodes

urgent channels

shared variables

selection bindings

quantifiers

channel arrays

committed nodes

broadcast channels

process priorities

Automation and Uppaal features

Why automate?

- Construction is tedious and error-prone,
- Testing reveals flaws which require fixes, then more testing. . .

Automate existing construction

[Stoelinga, 2002, Jensen et al., 2000]

But what about extra Uppaal features?

- urgent nodes ✓
- urgent channels ✓
- shared variables ✓
- selection bindings
- quantifiers
- channel arrays
- committed nodes
- broadcast channels
- process priorities

Automation and Uppaal features

Why automate?

- Construction is tedious and error-prone,
- Testing reveals flaws which require fixes, then more testing. . .

Automate existing construction

[Stoelinga, 2002, Jensen et al., 2000]

But what about extra Uppaal features?

urgent nodes	✓
urgent channels	✓
shared variables	✓
selection bindings	✓✓
quantifiers	✓✓
channel arrays	✓✓
committed nodes	
broadcast channels	
process priorities	

Automation and Uppaal features

Why automate?

- Construction is tedious and error-prone,
- Testing reveals flaws which require fixes, then more testing. . .

Automate existing construction

[Stoelinga, 2002, Jensen et al., 2000]

But what about extra Uppaal features?

urgent nodes	✓
urgent channels	✓
shared variables	✓
selection bindings	✓✓
quantifiers	✓✓
channel arrays	✓✓
committed nodes	✗
broadcast channels	
process priorities	

Automation and Uppaal features

Why automate?

- Construction is tedious and error-prone,
- Testing reveals flaws which require fixes, then more testing. . .

Automate existing construction

[Stoelinga, 2002, Jensen et al., 2000]

But what about extra Uppaal features?

urgent nodes	✓
urgent channels	✓
shared variables	✓
selection bindings	✓✓
quantifiers	✓✓
channel arrays	✓✓
committed nodes	✗
broadcast channels	✓ ✗
process priorities	

Automation and Uppaal features

Why automate?

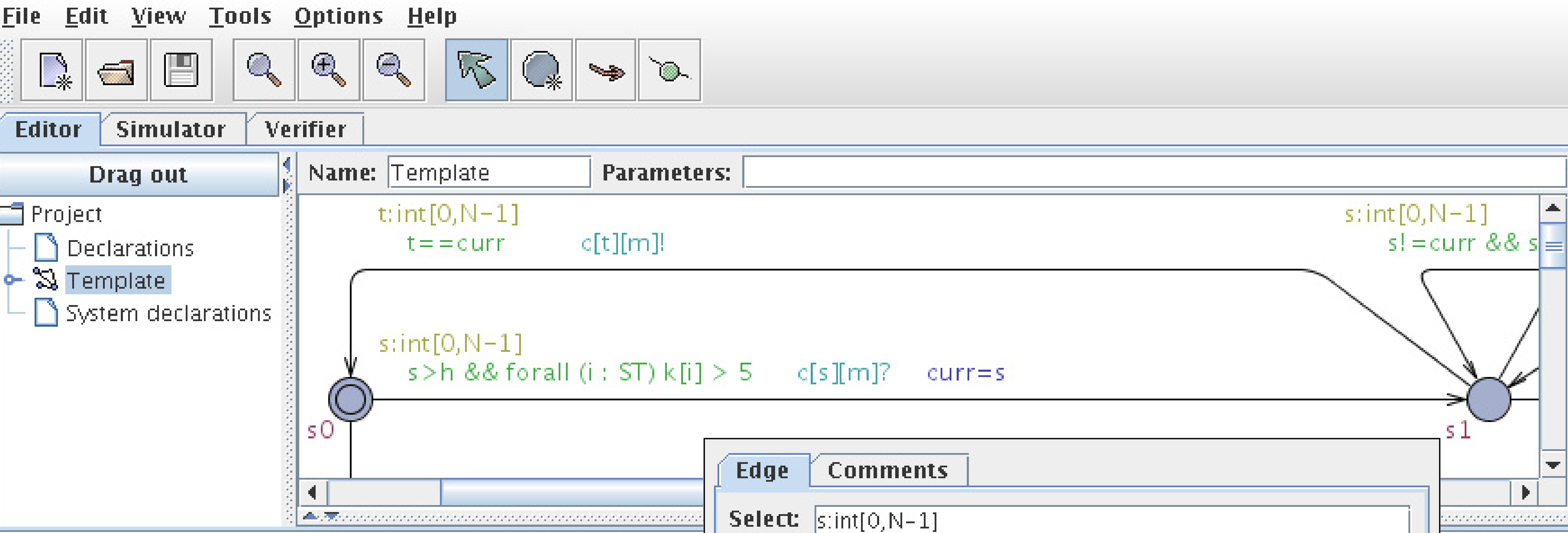
- Construction is tedious and error-prone,
- Testing reveals flaws which require fixes, then more testing...

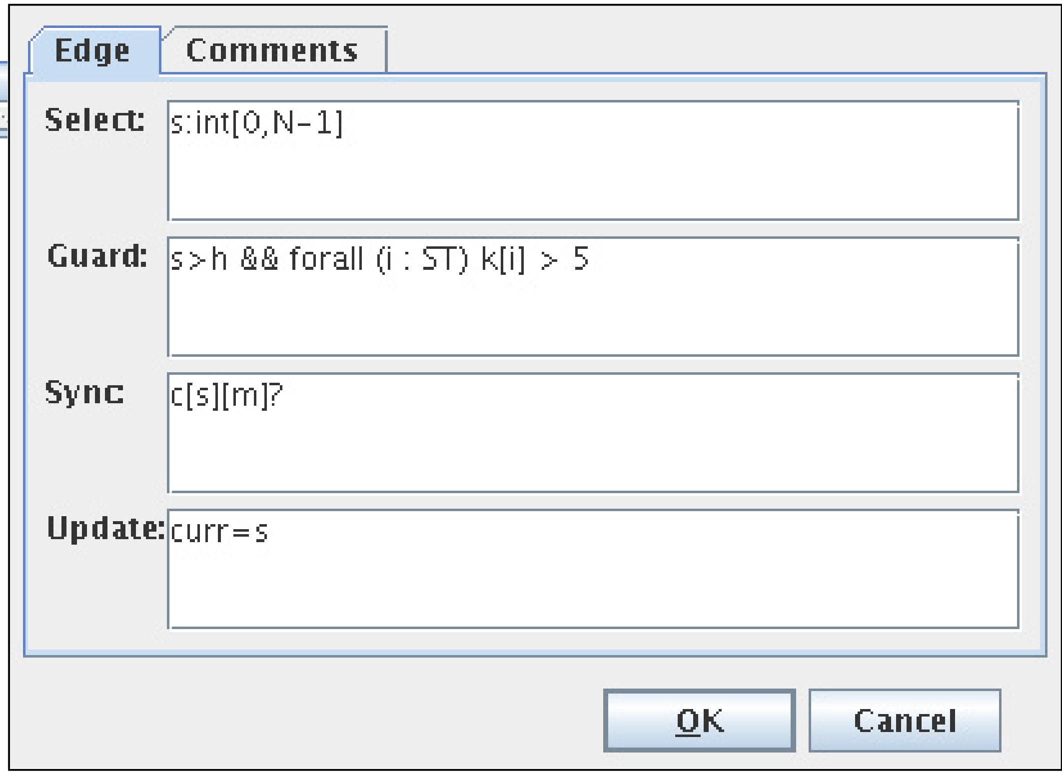
Automate existing construction

[Stoelinga, 2002, Jensen et al., 2000]

But what about extra Uppaal features?

urgent nodes	✓
urgent channels	✓
shared variables	✓
selection bindings	✓✓
quantifiers	✓✓
channel arrays	✓✓
committed nodes	✗
broadcast channels	✓ ✗
process priorities	✗





Uppaal transition features

- Basic guards
- Selection bindings
- Universal quantifiers in guards
- Channel arrays

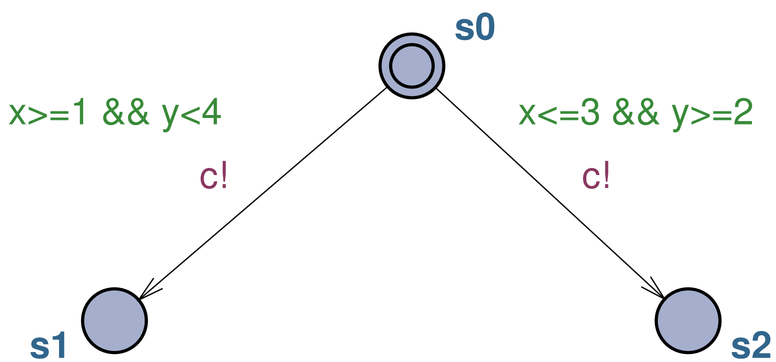
No selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join
2. Negate
3. DNF / Simplify

4. Split



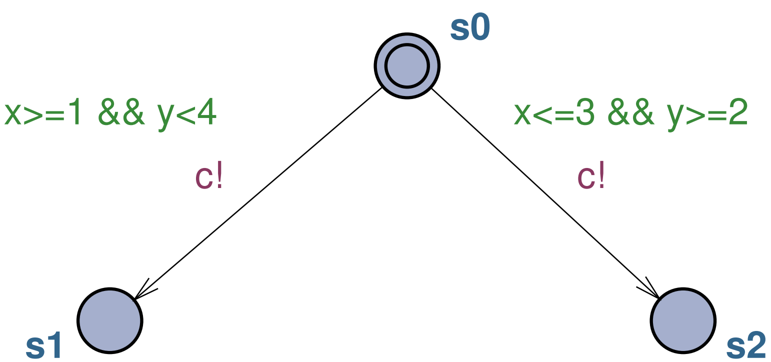
clock x, y;

No selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join $(x \geq 1 \wedge y < 4) \vee (x \leq 3 \wedge y \geq 2)$
2. Negate
3. DNF / Simplify
4. Split



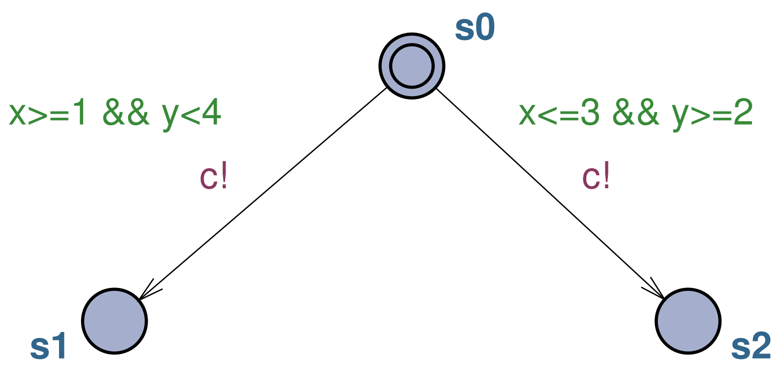
clock x, y;

No selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join $(x \geq 1 \wedge y < 4) \vee (x \leq 3 \wedge y \geq 2)$
2. Negate $(x < 1 \vee y \geq 4) \wedge (x > 3 \vee y < 2)$
3. DNF / Simplify
4. Split



clock x, y;

No selection bindings, No quantifiers, No channel arrays

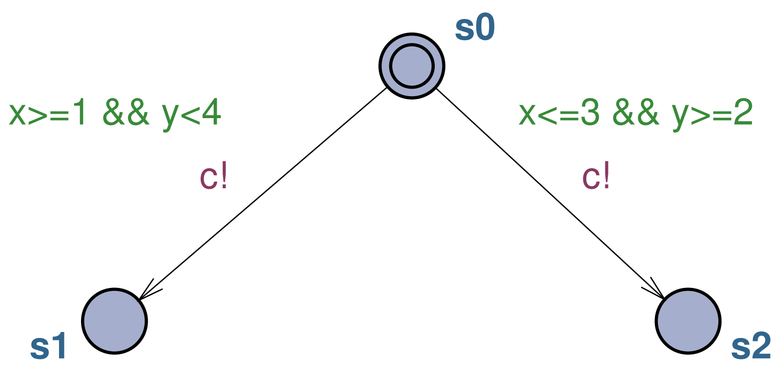
s0 c !

Group by state / channel / direction:

1. Join $(x \geq 1 \wedge y < 4) \vee (x \leq 3 \wedge y \geq 2)$
2. Negate $(x < 1 \vee y \geq 4) \wedge (x > 3 \vee y < 2)$
3. DNF / Simplify

$$(x < 1 \wedge x > 3) \vee (y \geq 4 \wedge x > 3) \vee (x < 1 \wedge y < 2) \vee (y \geq 4 \wedge y < 2)$$

4. Split



clock x, y;

No selection bindings, No quantifiers, No channel arrays

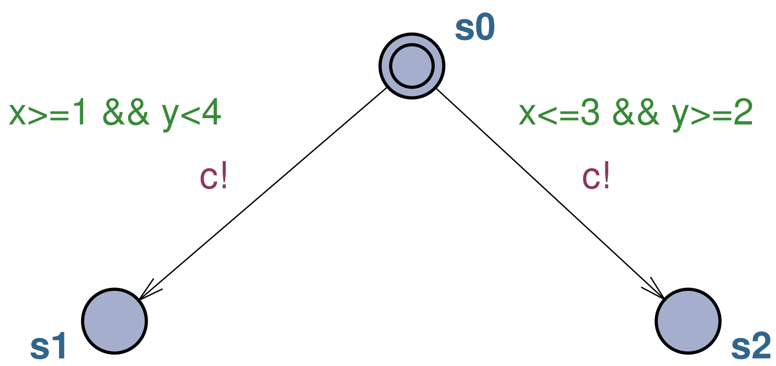
s0 c !

Group by state / channel / direction:

1. Join $(x \geq 1 \wedge y < 4) \vee (x \leq 3 \wedge y \geq 2)$
2. Negate $(x < 1 \vee y \geq 4) \wedge (x > 3 \vee y < 2)$
3. DNF / Simplify

~~$(x < 1 \wedge x > 3)$~~ $\vee \ (y \geq 4 \wedge x > 3) \ \vee \ (x < 1 \wedge y < 2) \ \vee$ ~~$(y \geq 4 \wedge y < 2)$~~

4. Split



clock x, y;

No selection bindings, No quantifiers, No channel arrays

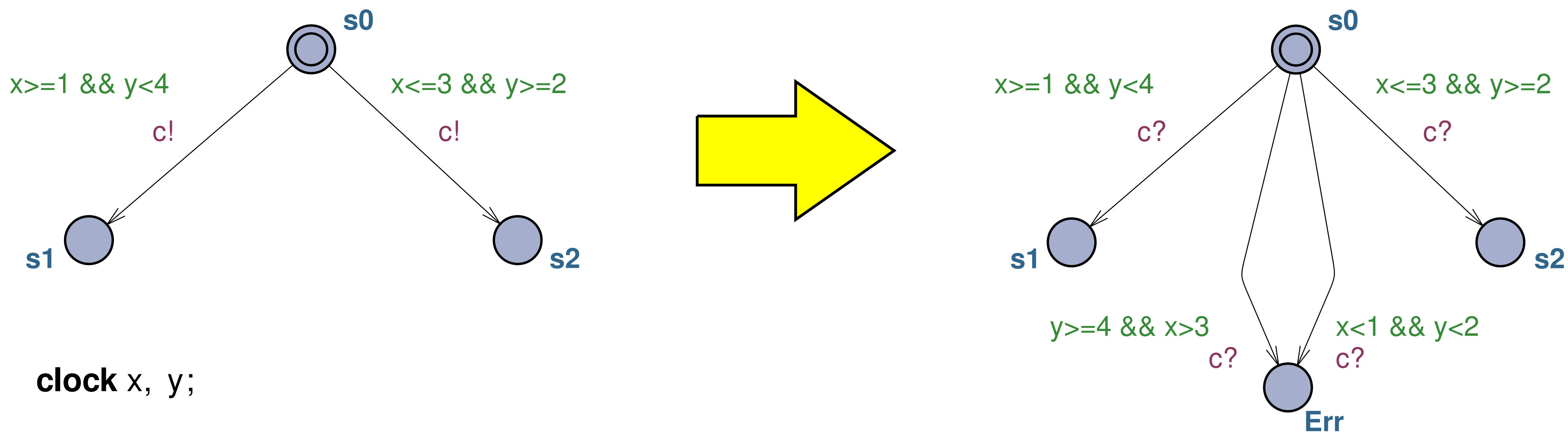
s0 c !

Group by state / channel / direction:

1. Join $(x \geq 1 \wedge y < 4) \vee (x \leq 3 \wedge y \geq 2)$
2. Negate $(x < 1 \vee y \geq 4) \wedge (x > 3 \vee y < 2)$
3. DNF / Simplify

$$(x < 1 \wedge x > 3) \vee (y \geq 4 \wedge x > 3) \vee (x < 1 \wedge y < 2) \vee (y \geq 4 \wedge y < 2)$$

4. Split



No selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

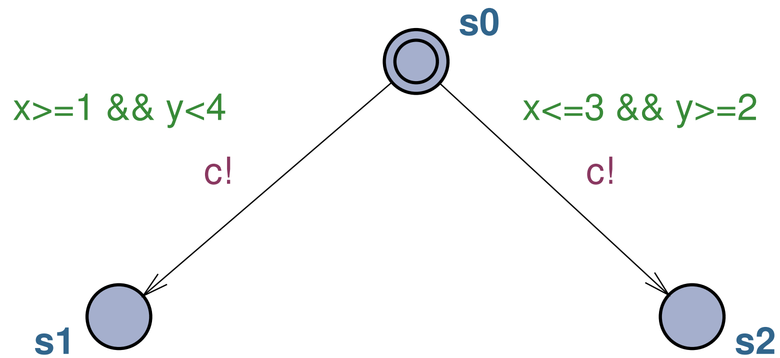
1. Join $(x \geq 1 \wedge y < 4) \vee (x \leq 3 \wedge y \geq 2)$

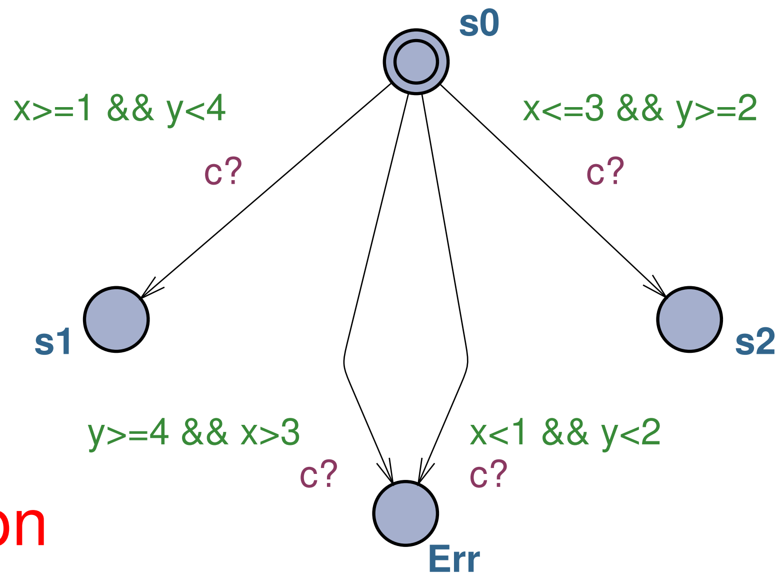
2. Negate $(x < 1 \vee y \geq 4) \wedge (x > 3 \vee y < 2)$

3. DNF / Simplify

$$(x < 1 \wedge x > 3) \vee (y \geq 4 \wedge x > 3) \vee (x < 1 \wedge y < 2) \vee (y \geq 4 \wedge y < 2)$$

4. Split





clock x, y;

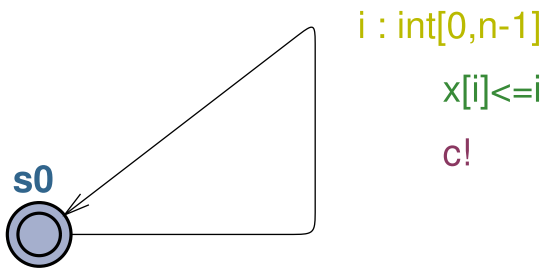
Key issue: splitting disjunction

Selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join
2. Negate
3. DNF / Simplify
4. Split



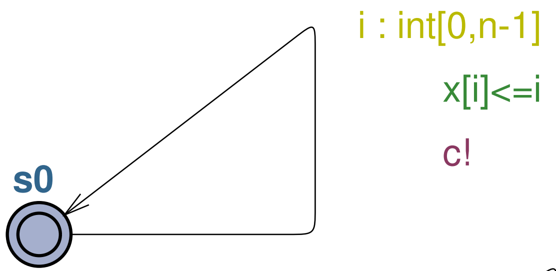
clock x[n];

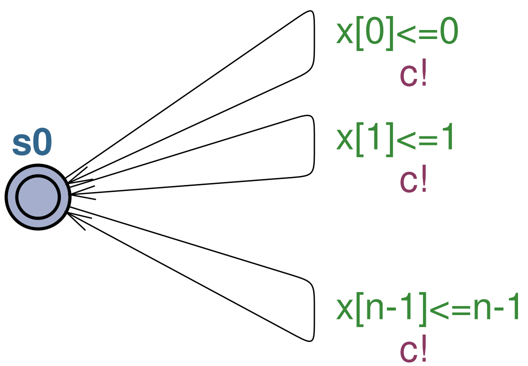
Selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join
2. Negate
3. DNF / Simplify
4. Split





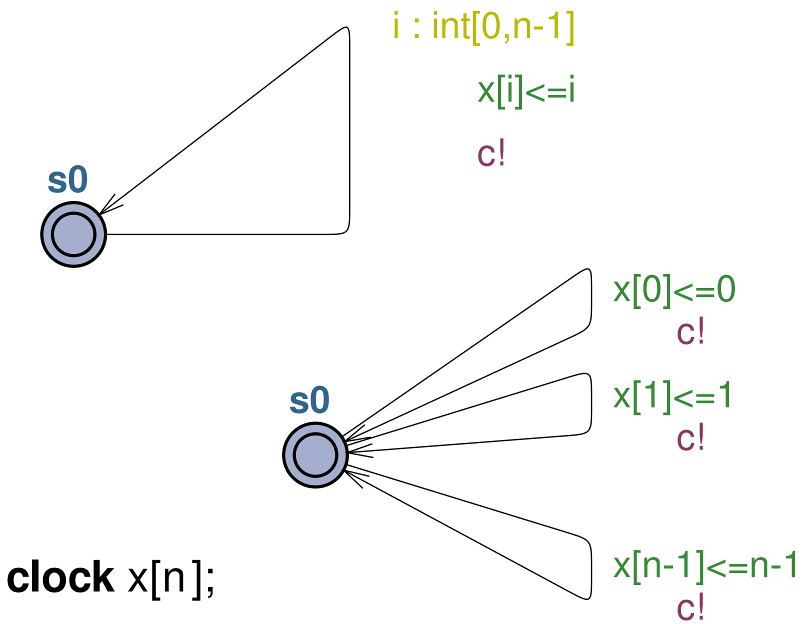
clock x[n];

Selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join $\exists i \in \{0, \ldots, n-1\}. x_i \leq i$
2. Negate
3. DNF / Simplify
4. Split

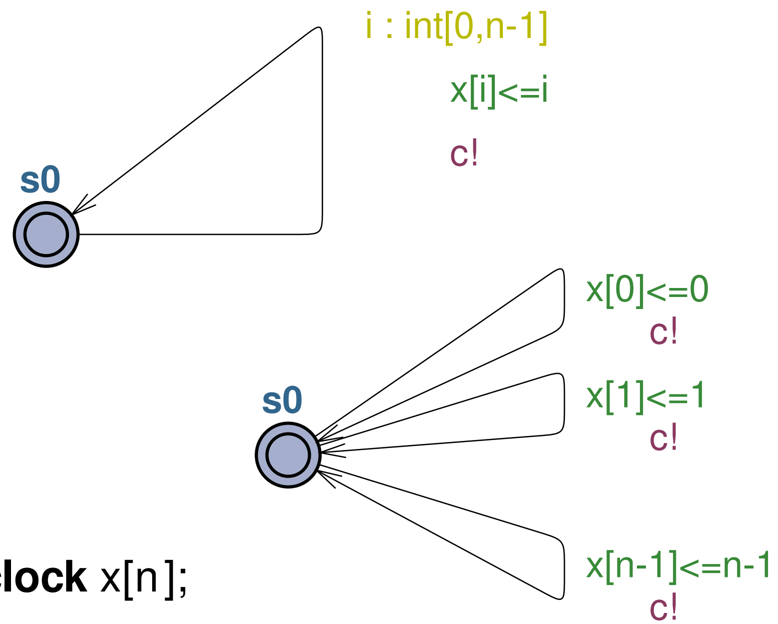


Selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join $\exists i \in \{0, \ldots, n-1\}.\, x_i \leq i$
2. Negate $\forall i \in \{0, \ldots, n-1\}.\, x_i > i$
3. DNF / Simplify
4. Split



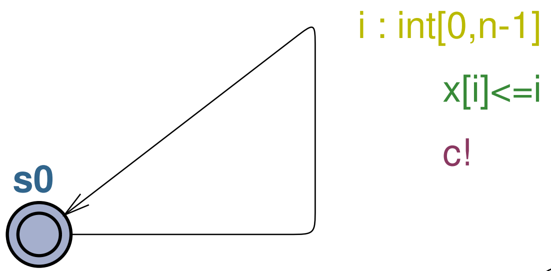
clock x[n];

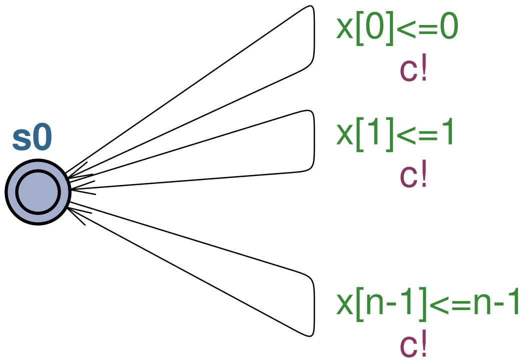
Selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join $\exists i \in \{0, \ldots, n-1\}.\, x_i \leq i$
2. Negate $\forall i \in \{0, \ldots, n-1\}.\, x_i > i$
3. DNF / Simplify $\forall i \in \{0, \ldots, n-1\}.\, x_i > i$
4. Split





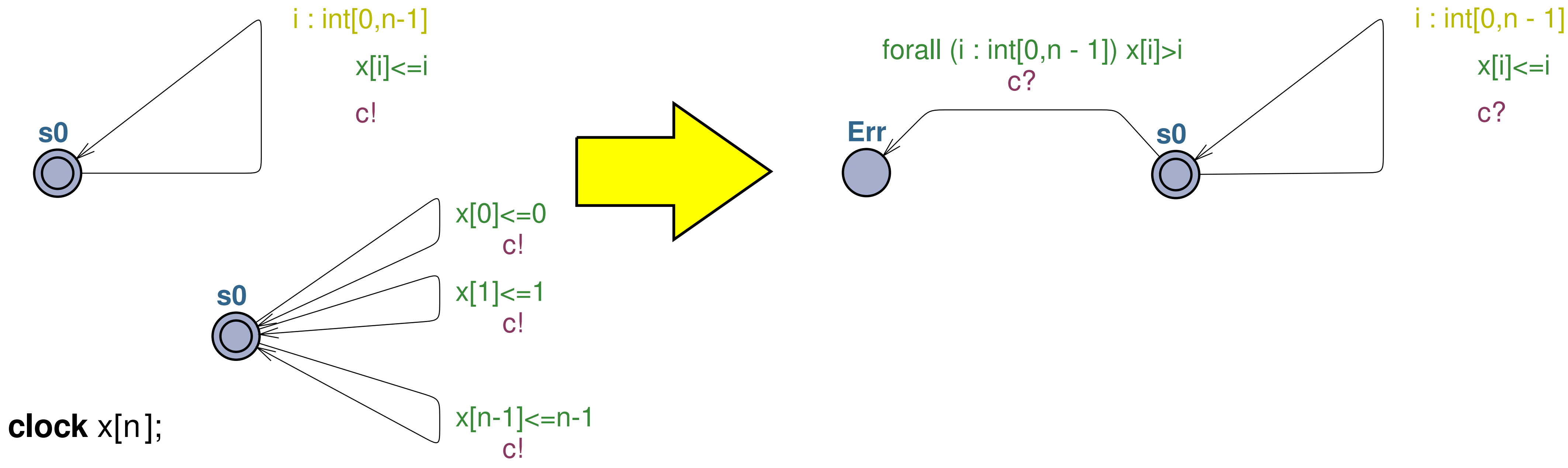
clock x[n];

Selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join $\exists i \in \{0,\ldots,n-1\}.\, x_i \leq i$
2. Negate $\forall i \in \{0,\ldots,n-1\}.\, x_i > i$
3. DNF / Simplify $\forall i \in \{0,\ldots,n-1\}.\, x_i > i$
4. Split

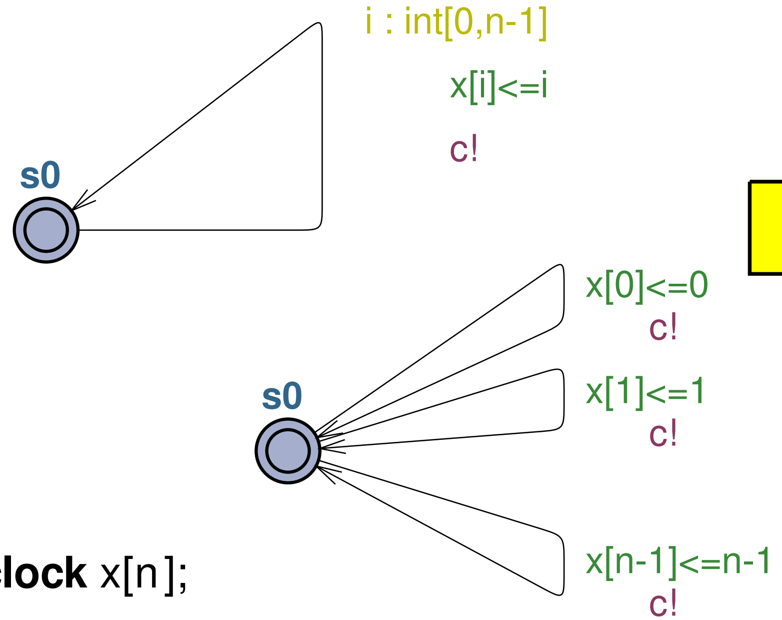


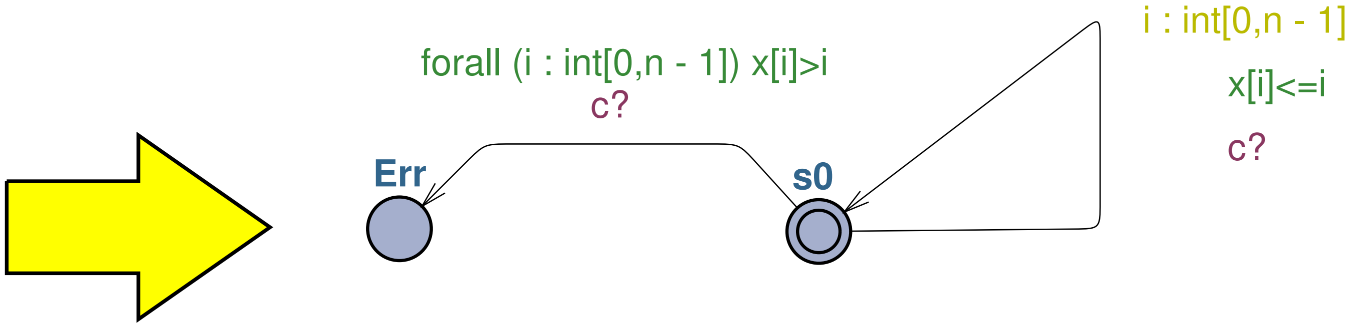
Selection bindings, No quantifiers, No channel arrays

s0 c !

Group by state / channel / direction:

1. Join $\exists i \in \{0, \ldots, n-1\}.\, x_i \leq i$
2. Negate $\forall i \in \{0, \ldots, n-1\}.\, x_i > i$
3. DNF / Simplify $\forall i \in \{0, \ldots, n-1\}.\, x_i > i$
4. Split





clock x[n];

Sometimes it's easy…

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join

2. Negate
3. DNF / Simplify
4. Split

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join $\quad (\exists s_{1\,1}, \ldots, s_{1\,n_1} \,.\, g_1) \vee \cdots \vee (\exists s_{m\,1}, \ldots, s_{m\,n_m} \,.\, g_m)$

2. Negate

3. DNF / Simplify

4. Split

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join $(\exists s_{1\,1}, \ldots, s_{1\,n_1} . g_1) \vee \cdots \vee (\exists s_{m\,1}, \ldots, s_{m\,n_m} . g_m)$

   $\exists s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . g_1 \vee \cdots \vee g_m$

2. Negate
3. DNF / Simplify
4. Split

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join $(\exists s_{1\,1}, \ldots, s_{1\,n_1} . g_1) \vee \cdots \vee (\exists s_{m\,1}, \ldots, s_{m\,n_m} . g_m)$

   $\exists s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . g_1 \vee \cdots \vee g_m$

2. Negate $\forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify

4. Split

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join $(\exists s_{1\,1}, \ldots, s_{1\,n_1} . g_1) \vee \cdots \vee (\exists s_{m\,1}, \ldots, s_{m\,n_m} . g_m)$

   $\exists s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . g_1 \vee \cdots \vee g_m$

2. Negate $\forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join $(\exists s_{1\,1}, \ldots, s_{1\,n_1} \,.\, g_1) \vee \cdots \vee (\exists s_{m\,1}, \ldots, s_{m\,n_m} \,.\, g_m)$

   $\exists s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} \,.\, g_1 \vee \cdots \vee g_m$

2. Negate $\forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} \,.\, \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} \,.\, \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split? $(\forall S'_1 \,.\, \overline{g}_1) \vee \cdots \vee (\forall S'_{m'} \,.\, \overline{g}_{m'})$

   not always possible!

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join $(\exists s_{1\,1}, \ldots, s_{1\,n_1} . g_1) \vee \cdots \vee (\exists s_{m\,1}, \ldots, s_{m\,n_m} . g_m)$

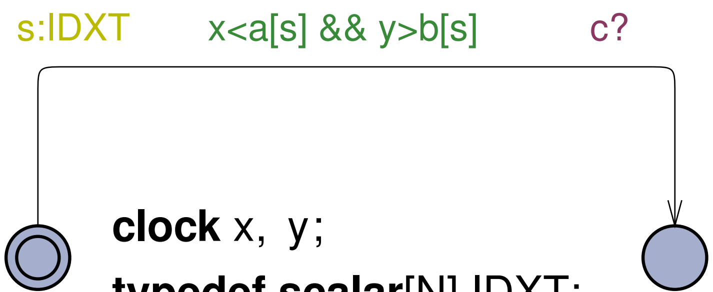
   $\exists s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . g_1 \vee \cdots \vee g_m$

2. Negate $\forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} . \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split**?** $(\forall S'_1 . \overline{g}_1) \vee \cdots \vee (\forall S'_{m'} . \overline{g}_{m'})$

not always possible!



clock x, y;
typedef scalar[N] IDXT;
int a[IDXT], b[IDXT];

Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \ldots, (S_m, g_m)\}$$

1. Join $\quad (\exists s_{1\,1}, \ldots, s_{1\,n_1} \,.\, g_1) \vee \cdots \vee (\exists s_{m\,1}, \ldots, s_{m\,n_m} \,.\, g_m)$

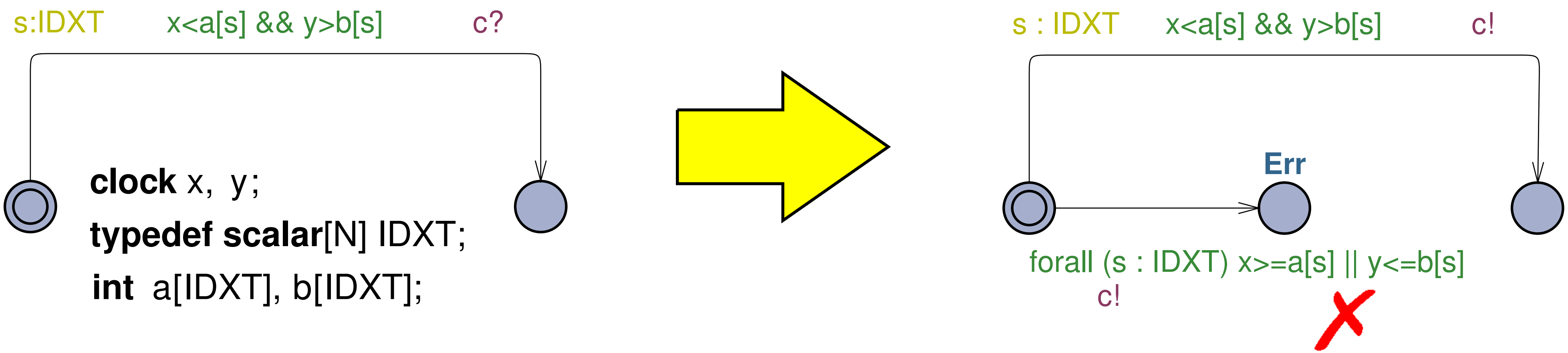
   $\exists s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} \,.\, g_1 \vee \cdots \vee g_m$

2. Negate $\quad \forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} \,.\, \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\quad \forall s_{1\,1}, \ldots, s_{1\,n_1}, \ldots, s_{m\,1}, \ldots, s_{m\,n_m} \,.\, \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split**?** $\quad (\forall S'_1 \,.\, \overline{g}_1) \vee \cdots \vee (\forall S'_{m'} \,.\, \overline{g}_{m'})$

not always possible!



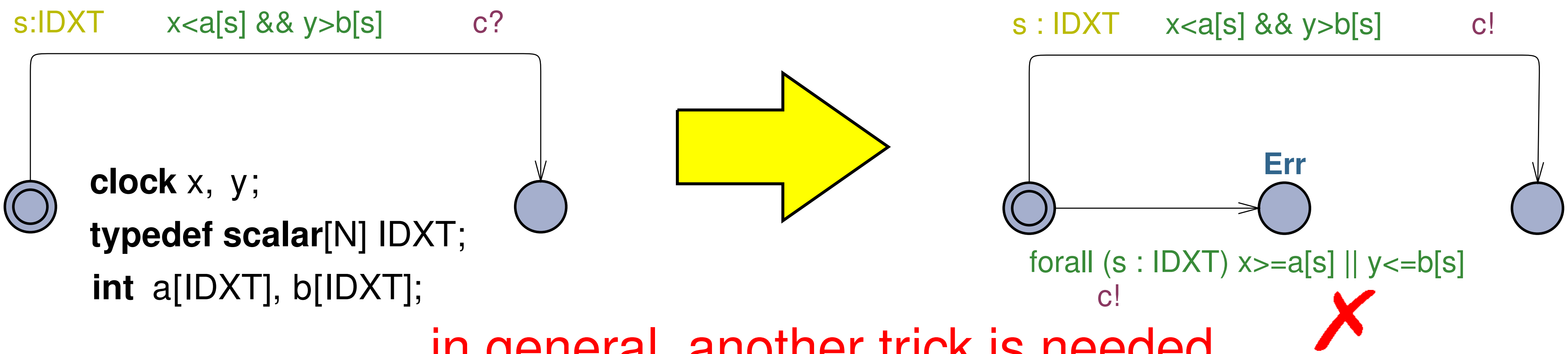
Selection bindings, No quantifiers, No channel arrays

$$E = \{(S_1, g_1), \dots, (S_m, g_m)\}$$

1. Join $(\exists s_{1\,1}, \dots, s_{1\,n_1} \cdot g_1) \vee \dots \vee (\exists s_{m\,1}, \dots, s_{m\,n_m} \cdot g_m)$

   $\exists s_{1\,1}, \dots, s_{1\,n_1}, \dots, s_{m\,1}, \dots, s_{m\,n_m} \cdot g_1 \vee \dots \vee g_m$
2. Negate $\forall s_{1\,1}, \dots, s_{1\,n_1}, \dots, s_{m\,1}, \dots, s_{m\,n_m} \cdot \neg g_1 \wedge \dots \wedge \neg g_m$
3. DNF / Simplify $\forall s_{1\,1}, \dots, s_{1\,n_1}, \dots, s_{m\,1}, \dots, s_{m\,n_m} \cdot \overline{g}_1 \vee \dots \vee \overline{g}_{m'}$
4. Split? $(\forall S'_1 \cdot \overline{g}_1) \vee \dots \vee (\forall S'_{m'} \cdot \overline{g}_{m'})$

not always possible!



... in general, another trick is needed.

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join

2. Negate
3. DNF / Simplify
4. Split

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join $(\exists S_1\, \forall A_1 .\, g_1) \vee \cdots \vee (\exists S_m\, \forall A_m .\, g_m)$
2. Negate
3. DNF / Simplify
4. Split

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join

$$(\exists S_1 \, \forall A_1 \, . \, g_1) \vee \cdots \vee (\exists S_m \, \forall A_m \, . \, g_m)$$

$$\exists S_1, \ldots, S_m \, \forall A_1, \ldots, A_m \, . \, g_1 \vee \cdots \vee g_m$$

2. Negate
3. DNF / Simplify
4. Split

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join $(\exists S_1\, \forall A_1 .\, g_1) \vee \cdots \vee (\exists S_m\, \forall A_m .\, g_m)$

   $\exists S_1, \ldots, S_m\, \forall A_1, \ldots, A_m .\, g_1 \vee \cdots \vee g_m$
2. Negate $\forall S_1, \ldots, S_m\, \exists A_1, \ldots, A_m .\, \neg g_1 \wedge \cdots \wedge \neg g_m$
3. DNF / Simplify
4. Split

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join $(\exists S_1 \, \forall A_1 \, . \, g_1) \vee \cdots \vee (\exists S_m \, \forall A_m \, . \, g_m)$

   $\exists S_1, \ldots, S_m \, \forall A_1, \ldots, A_m \, . \, g_1 \vee \cdots \vee g_m$

2. Negate $\forall S_1, \ldots, S_m \, \exists A_1, \ldots, A_m \, . \, \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall S_1, \ldots, S_m \, \exists A_1, \ldots, A_m \, . \, \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join $(\exists S_1\ \forall A_1 \,.\, g_1) \vee \cdots \vee (\exists S_m\ \forall A_m \,.\, g_m)$

   $\exists S_1, \ldots, S_m\ \forall A_1, \ldots, A_m \,.\, g_1 \vee \cdots \vee g_m$

2. Negate $\forall S_1, \ldots, S_m\ \exists A_1, \ldots, A_m \,.\, \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall S_1, \ldots, S_m\ \exists A_1, \ldots, A_m \,.\, \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join $(\exists S_1\ \forall A_1 .\, g_1) \vee \cdots \vee (\exists S_m\ \forall A_m .\, g_m)$

   $\exists S_1, \ldots, S_m\ \forall A_1, \ldots, A_m .\, g_1 \vee \cdots \vee g_m$

2. Negate $\forall S_1, \ldots, S_m\ \exists A_1, \ldots, A_m .\, \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall S_1, \ldots, S_m\ \exists A_1, \ldots, A_m .\, \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split? $\exists A_1, \ldots, A_m .\, (\forall S'_1 .\, \overline{g}_1) \vee \cdots \vee (\forall S'_{m'} .\, \overline{g}_{m'})$

even worse!

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join $(\exists S_1\, \forall A_1 .\, g_1) \vee \cdots \vee (\exists S_m\, \forall A_m .\, g_m)$

   $\exists S_1, \ldots, S_m\, \forall A_1, \ldots, A_m .\, g_1 \vee \cdots \vee g_m$

2. Negate $\forall S_1, \ldots, S_m\, \exists A_1, \ldots, A_m .\, \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall S_1, \ldots, S_m\, \exists A_1, \ldots, A_m .\, \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split? $\exists A_1, \ldots, A_m .\, (\forall S'_1 .\, \overline{g}_1) \vee \cdots \vee (\forall S'_{m'} .\, \overline{g}_{m'})$

even worse!

- Devise a predicate canswap(φ)
- Use a looping construction (if no scalars)

  (also works for simpler case where $A_i = \emptyset$)

Selection bindings, Quantifiers, No channel arrays

$$E = \{(S_1, A_1, g_1), \ldots, (S_m, A_m, g_m)\}$$

1. Join $(\exists S_1\ \forall A_1 .\ g_1) \vee \cdots \vee (\exists S_m\ \forall A_m .\ g_m)$

   $\exists S_1, \ldots, S_m\ \forall A_1, \ldots, A_m .\ g_1 \vee \cdots \vee g_m$

2. Negate $\forall S_1, \ldots, S_m\ \exists A_1, \ldots, A_m .\ \neg g_1 \wedge \cdots \wedge \neg g_m$

3. DNF / Simplify $\forall S_1, \ldots, S_m\ \exists A_1, \ldots, A_m .\ \overline{g}_1 \vee \cdots \vee \overline{g}_{m'}$

4. Split? $\exists A_1, \ldots, A_m .\ (\forall S'_1 .\ \overline{g}_1) \vee \cdots \vee (\forall S'_{m'} .\ \overline{g}_{m'})$

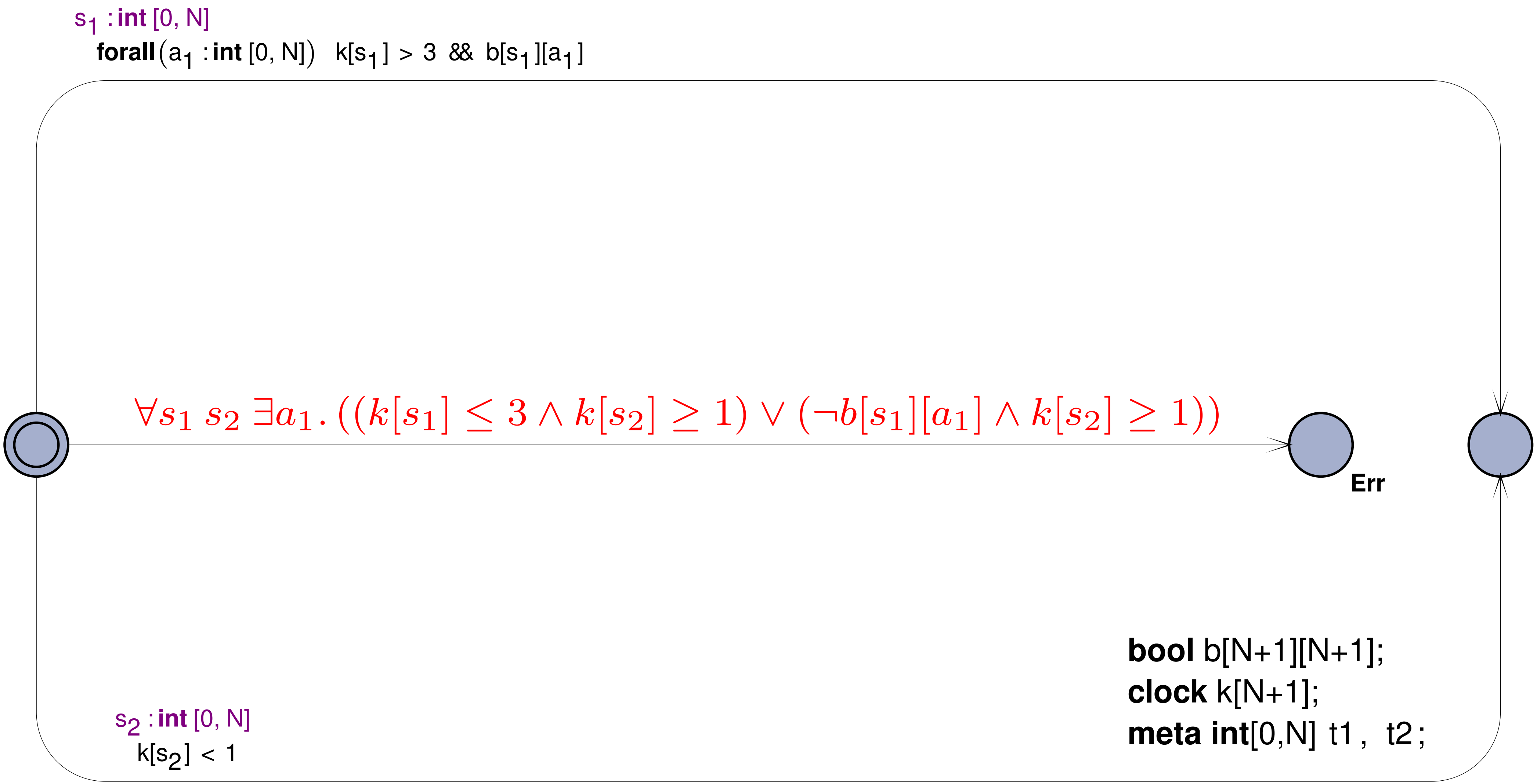
even worse!

- Devise a predicate canswap(φ) ✓
- Use a looping construction (if no scalars) (not yet)

  (also works for simpler case where $A_i = \emptyset$)

Selection bindings, Quantifiers, No channel arrays

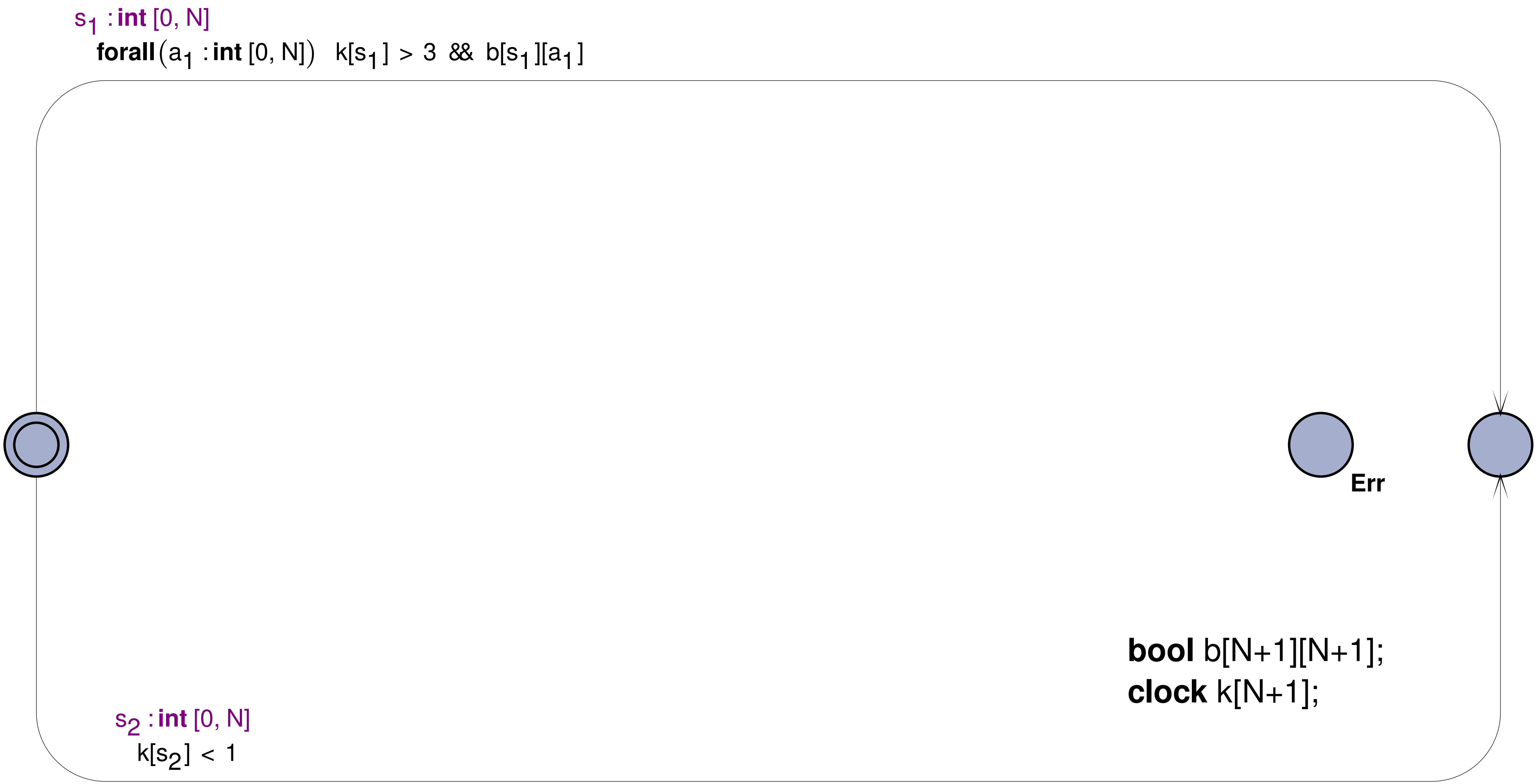
$$\exists s_1\, s_2\, \forall a_1 .\, ((k[s_1] > 3 \wedge b[s_1][a_1]) \vee (k[s_2] < 1))$$



Selection bindings, Quantifiers, No channel arrays

$$\exists s_1\ s_2\ \forall a_1 .\ ((k[s_1] > 3 \land b[s_1][a_1]) \lor (k[s_2] < 1))$$

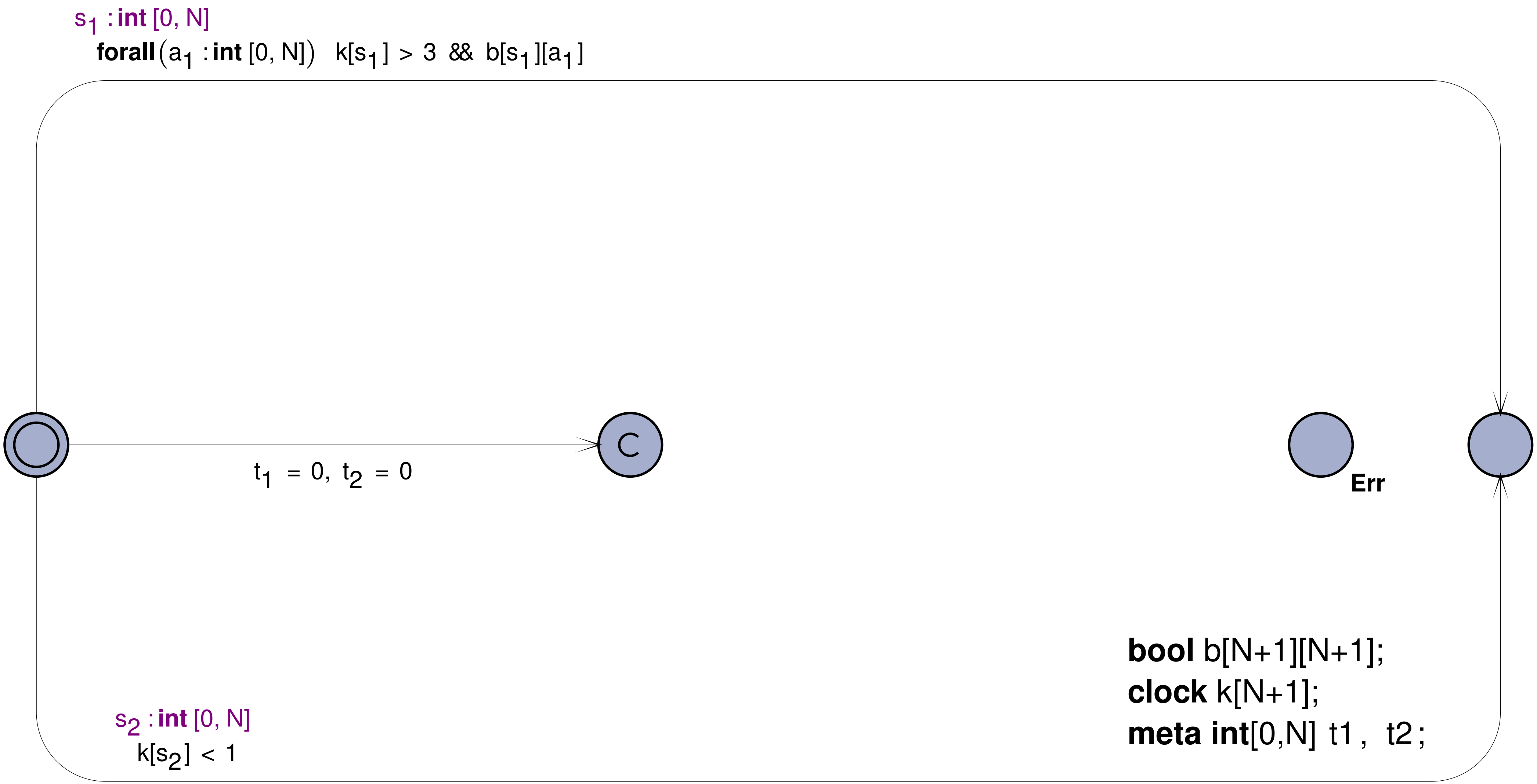
$$\forall s_1\ s_2\ \exists a_1 .\ ((k[s_1] \leq 3 \land k[s_2] \geq 1) \lor (\neg b[s_1][a_1] \land k[s_2] \geq 1))$$



Selection bindings, Quantifiers, No channel arrays

$$\exists s_1\, s_2\, \forall a_1 .\, ((k[s_1] > 3 \wedge b[s_1][a_1]) \vee (k[s_2] < 1))$$

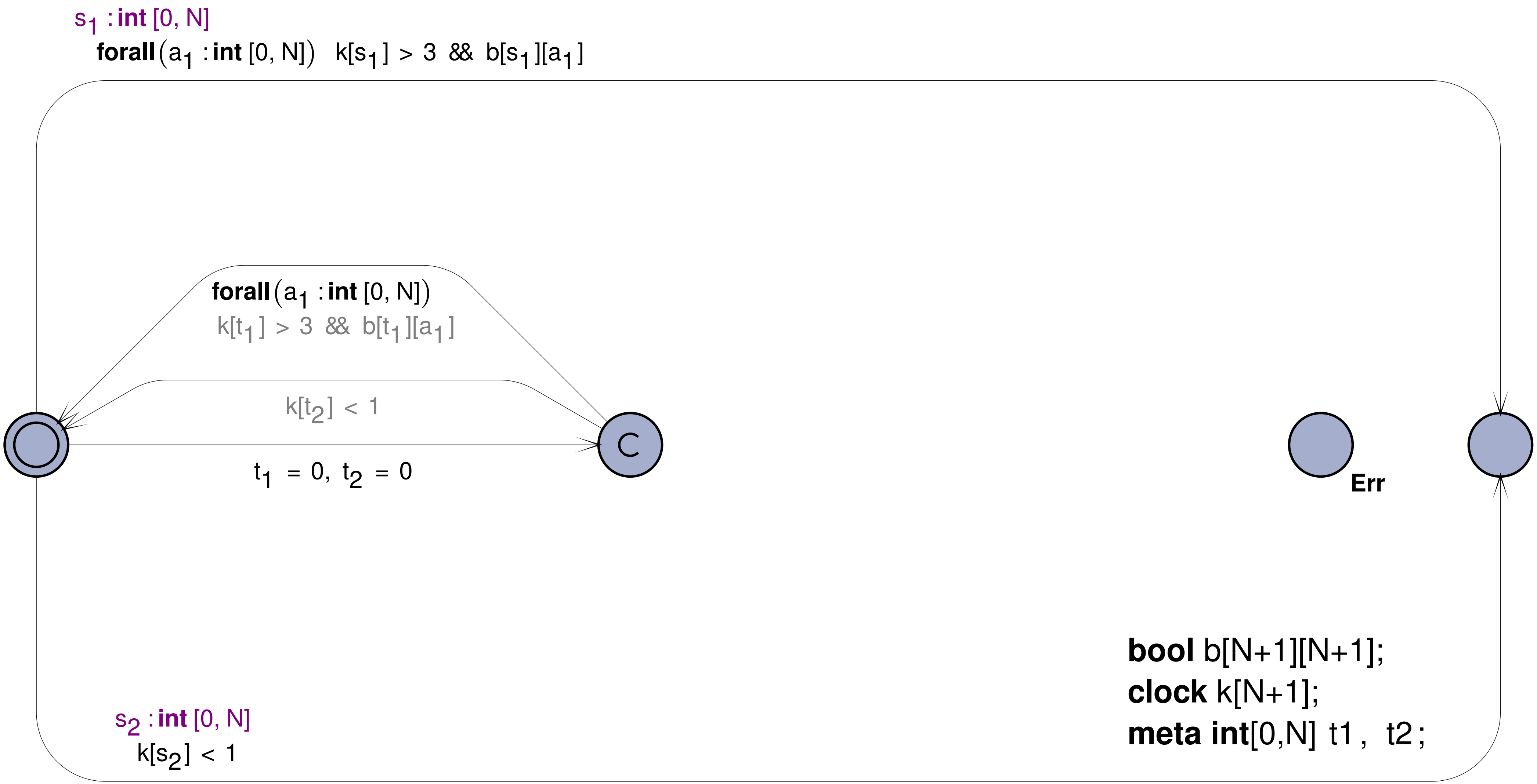
$$\forall s_1\, s_2\, \exists a_1 .\, ((k[s_1] \leq 3 \wedge k[s_2] \geq 1) \vee (\neg b[s_1][a_1] \wedge k[s_2] \geq 1))$$



Selection bindings, Quantifiers, No channel arrays

$$\exists s_1\ s_2\ \forall a_1 .\ ((k[s_1] > 3 \wedge b[s_1][a_1]) \vee (k[s_2] < 1))$$

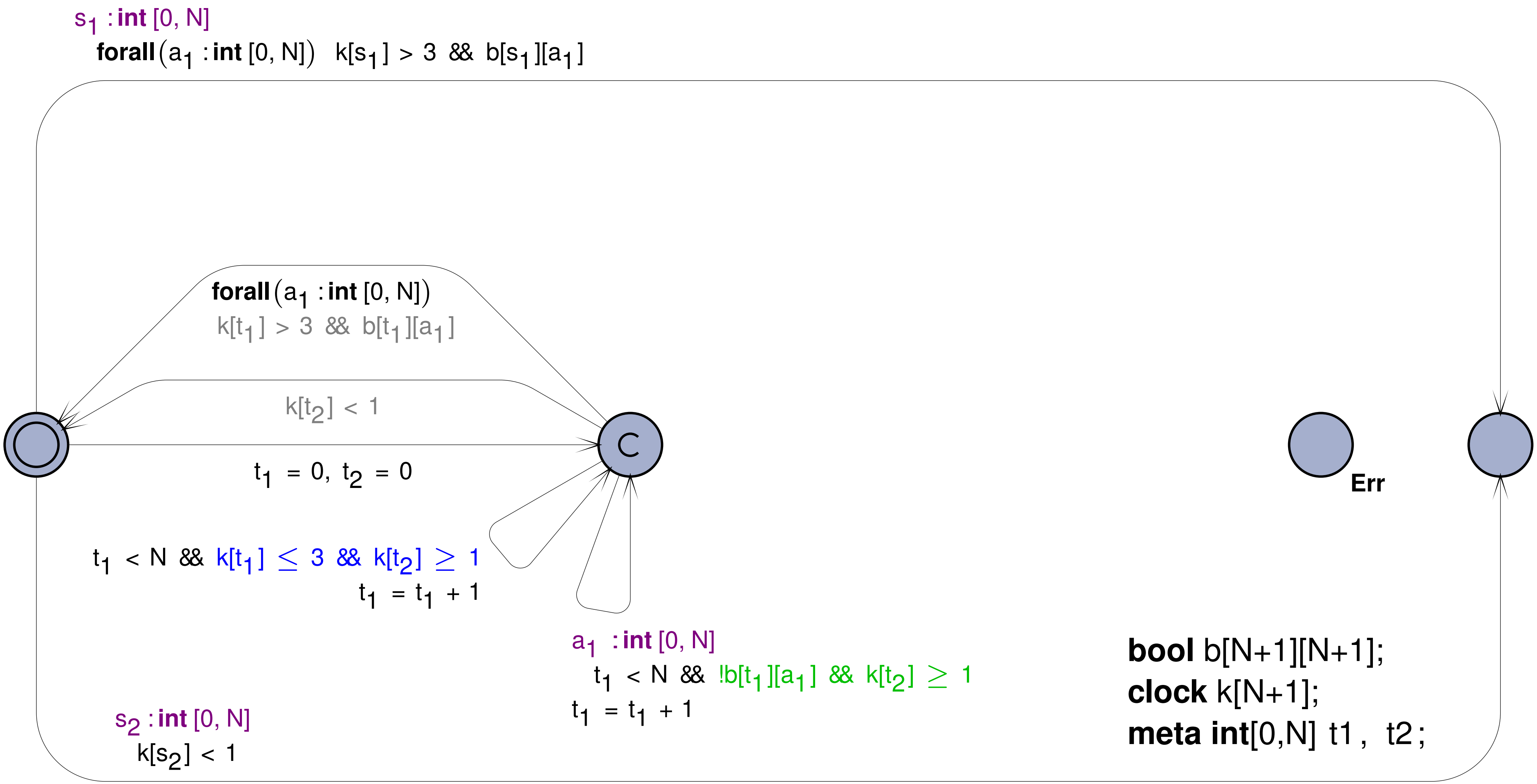
$$\forall s_1\ s_2\ \exists a_1 .\ ((k[s_1] \leq 3 \wedge k[s_2] \geq 1) \vee (\neg b[s_1][a_1] \wedge k[s_2] \geq 1))$$



Selection bindings, Quantifiers, No channel arrays

$$\exists s_1\ s_2\ \forall a_1 .\ ((k[s_1] > 3 \wedge b[s_1][a_1]) \vee (k[s_2] < 1))$$

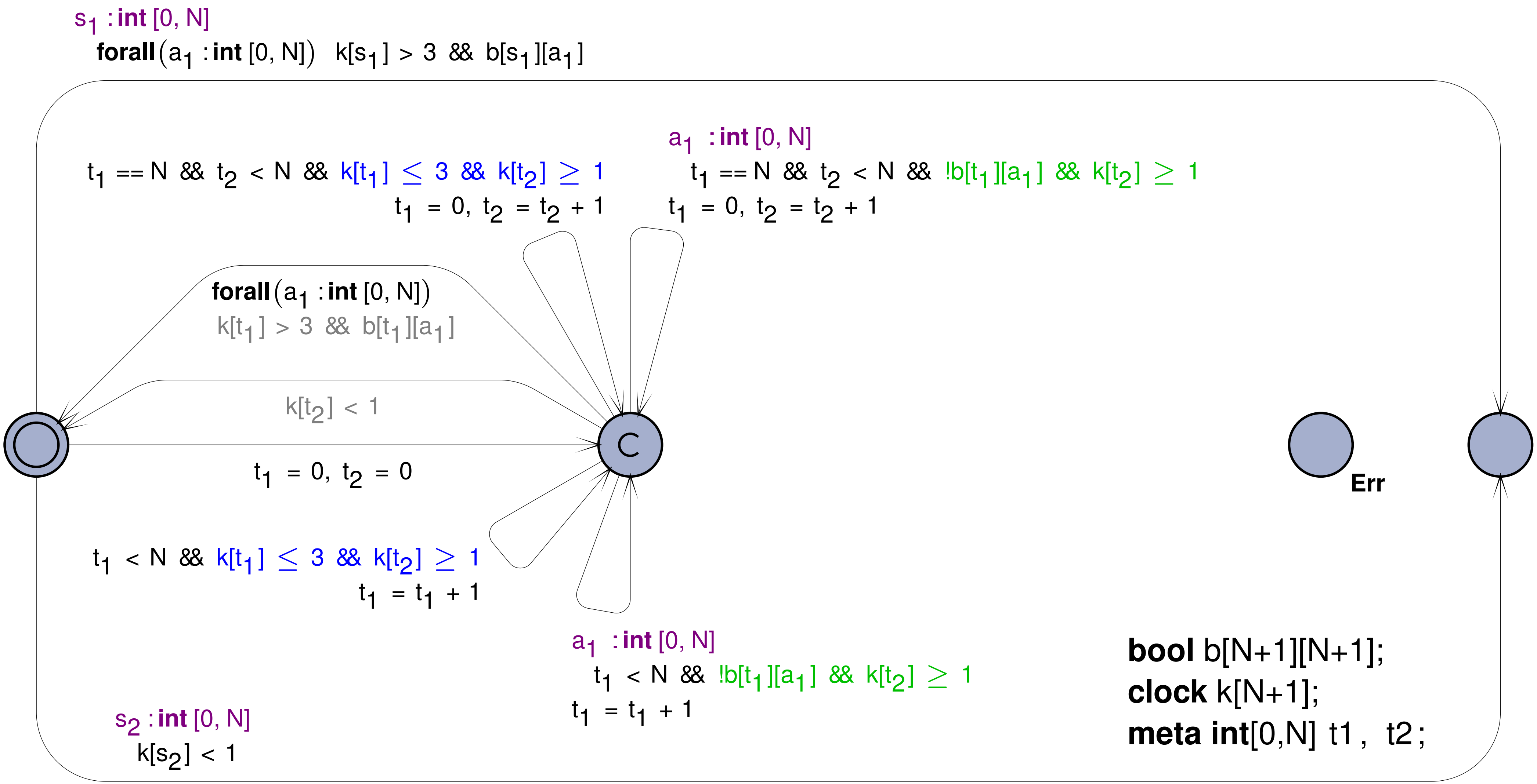
$$\forall s_1\ s_2\ \exists a_1 .\ ((k[s_1] \leq 3 \wedge k[s_2] \geq 1) \vee (\neg b[s_1][a_1] \wedge k[s_2] \geq 1))$$



Selection bindings, Quantifiers, No channel arrays

$$\exists s_1\ s_2\ \forall a_1 .\ ((k[s_1] > 3 \land b[s_1][a_1]) \lor (k[s_2] < 1))$$

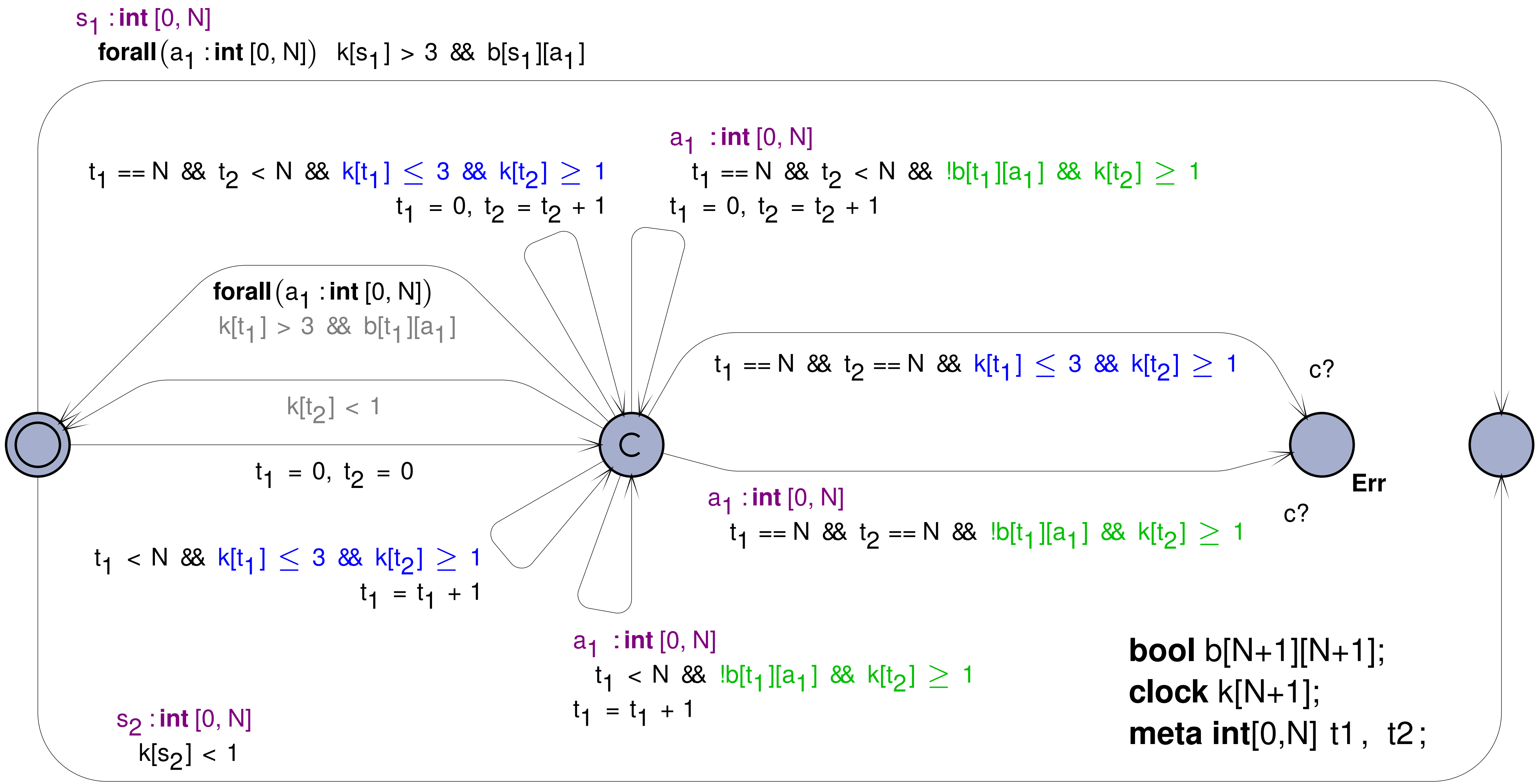
$$\forall s_1\ s_2\ \exists a_1 .\ ((k[s_1] \leq 3 \land k[s_2] \geq 1) \lor (\neg b[s_1][a_1] \land k[s_2] \geq 1))$$



Selection bindings, Quantifiers, No channel arrays

$$\exists s_1\, s_2\, \forall a_1 .\, ((k[s_1] > 3 \land b[s_1][a_1]) \lor (k[s_2] < 1))$$

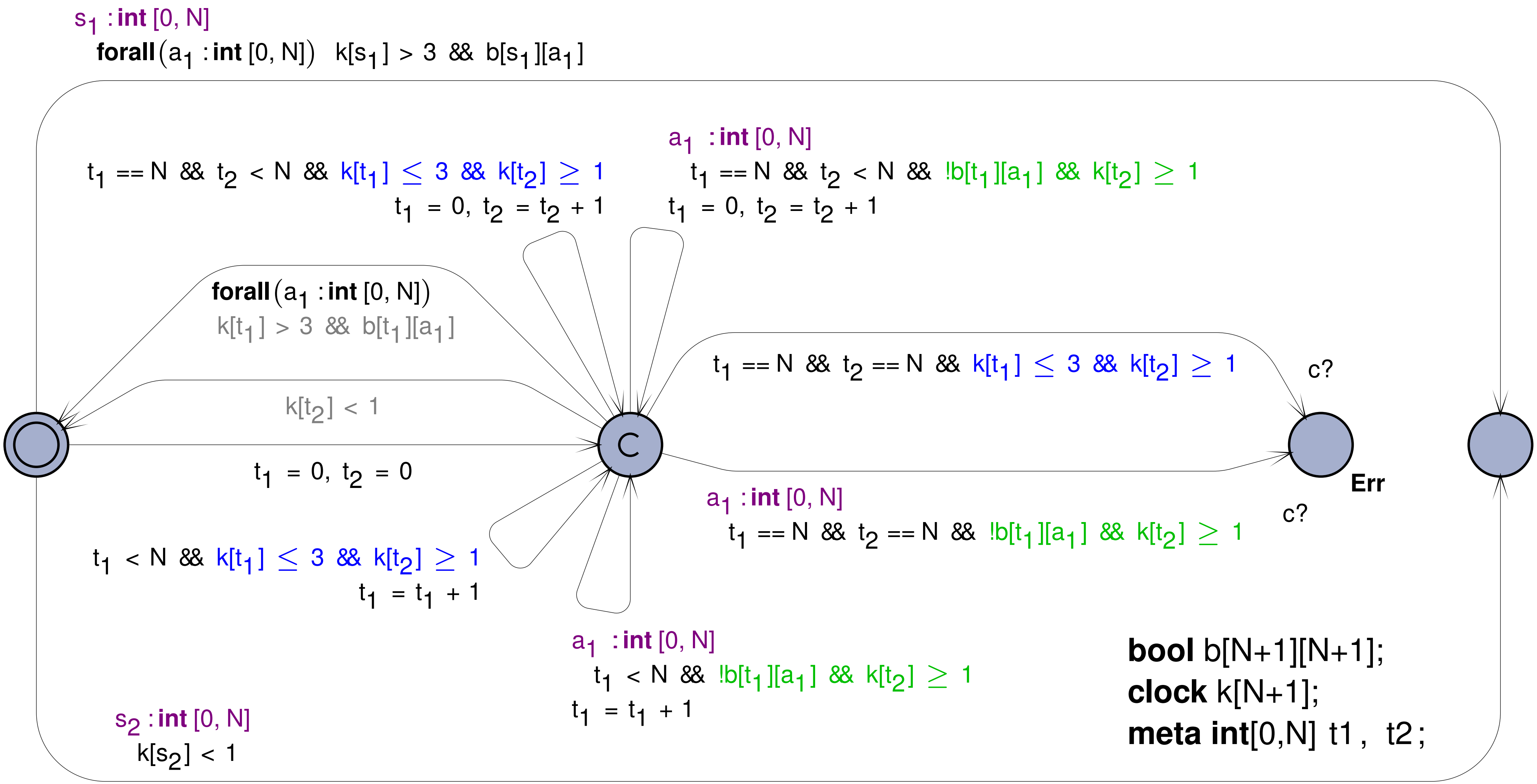
$$\forall s_1\, s_2\, \exists a_1 .\, ((k[s_1] \leq 3 \land k[s_2] \geq 1) \lor (\neg b[s_1][a_1] \land k[s_2] \geq 1))$$



Selection bindings, Quantifiers, No channel arrays

$$\exists s_1\ s_2\ \forall a_1 .\ ((k[s_1] > 3 \land b[s_1][a_1]) \lor (k[s_2] < 1))$$

$$\forall s_1\ s_2\ \exists a_1 .\ ((k[s_1] \leq 3 \land k[s_2] \geq 1) \lor (\neg b[s_1][a_1] \land k[s_2] \geq 1))$$



Conjecture that this always works (for bounded integers)

Channel arrays, No Selection bindings

bounded integers

scalars

chan c[N][ST];

multidimensional

Channel arrays, No Selection bindings

Group by state / channel / direction

bounded integers

scalars

chan c[N][ST];

multidimensional

Channel arrays, No Selection bindings

s0 !

Group by state / channel / direction

c is a set of channels

chan c[N][ST];

bounded integers

scalars

multidimensional

Channel arrays, No Selection bindings

s0 !

Group by state / channel / direction

c is a set of channels

bounded integers

scalars

chan c[N][ST];

multidimensional

e.g. c e.g. [2∗i][s][3]

Synchronisations specify an element of a set by a sequence of expressions

Channel arrays, No Selection bindings

s0 !

Group by state / channel / direction

c is a set of channels

chan c[N][ST];

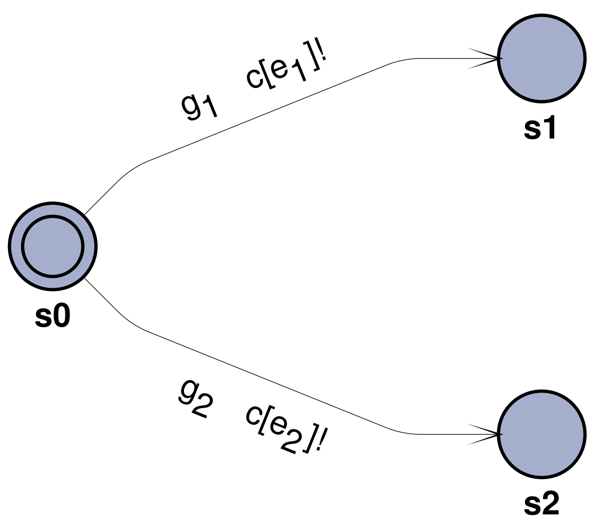
bounded integers

scalars

multidimensional

e.g. c e.g. [2∗i][s][3]

Synchronisations specify an element of a set by a sequence of expressions



Channel arrays, No Selection bindings

s0 !

Group by state / channel / direction

c is a set of channels

bounded integers

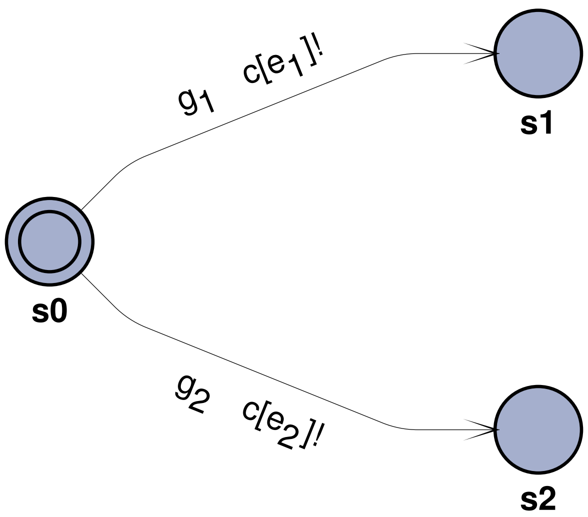
scalars

chan c[N][ST];

multidimensional

e.g. c e.g. [2∗i][s][3]

Synchronisations specify an element of a set by a sequence of expressions



Two possible groupings:

$e_1 = e_2$ negate $g_1 \vee g_2$

cover other channels

$e_1 \neq e_2$ negate g_1

negate g_2

cover other channels

Channel arrays, (some) Selection bindings

$$E = \left\{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1\rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m\rangle)\right\}$$

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1\rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m\rangle)\}$$

e_i
- expression over state variables
- single selection binding over whole range

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1 \rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m \rangle)\}$$

e_i

- expression over state variables
- single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3, 5]\}, A, g, \langle s \rangle)$

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1 \rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m \rangle)\}$$

e_i

- expression over state variables
- single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3, 5]\}, A, g, \langle s \rangle)$

$\Longrightarrow (\{s : [0, 8]\}, A, g \wedge (s \geq 3) \wedge (s \leq 5), \langle s \rangle)$

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1\rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m\rangle)\}$$

e_i
- expression over state variables
- single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3, 5]\}, A, g, \langle s\rangle)$

$\Longrightarrow (\{s : [0, 8]\}, A, g \wedge (s \geq 3) \wedge (s \leq 5), \langle s\rangle)$

- It can be done.

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1 \rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m \rangle)\}$$

e_i
- expression over state variables
- single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3, 5]\}, A, g, \langle s \rangle)$

$\Longrightarrow (\{s : [0, 8]\}, A, g \wedge (s \geq 3) \wedge (s \leq 5), \langle s \rangle)$

- It can be done.
- No detail in this presentation!

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1 \rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m \rangle)\}$$

e_i — expression over state variables

e_i — single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3, 5]\}, A, g, \langle s \rangle)$

$\Longrightarrow (\{s : [0, 8]\}, A, g \wedge (s \geq 3) \wedge (s \leq 5), \langle s \rangle)$

- It can be done.
- No detail in this presentation!
- Introduce selection bindings to cover all channels...
- ...add a predicate to each transition before joining them.

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e^1_1, \ldots, e^1_{n_C} \rangle), \ldots, (S_m, A_m, g_m, \langle e^m_1, \ldots, e^m_{n_C} \rangle)\}$$

e_i
- expression over state variables
- single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3, 5]\}, A, g, \langle s \rangle)$

$\Longrightarrow (\{s : [0, 8]\}, A, g \wedge (s \geq 3) \wedge (s \leq 5), \langle s \rangle)$

- It can be done.
- No detail in this presentation!
- Introduce selection bindings to cover all channels…
- …add a predicate to each transition before joining them.

What about more general expressions involving selection bindings?

Key property: each S_w valuation specifies a different channel

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e^1_1, \ldots, e^1_{n_C}\rangle), \ldots, (S_m, A_m, g_m, \langle e^m_1, \ldots, e^m_{n_C}\rangle)\}$$

e_i — expression over state variables

e_i — single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3, 5]\}, A, g, \langle s \rangle)$

$\Longrightarrow (\{s : [0, 8]\}, A, g \wedge (s \geq 3) \wedge (s \leq 5), \langle s \rangle)$

- It can be done.
- No detail in this presentation!
- Introduce selection bindings to cover all channels. . .
- . . . add a predicate to each transition before joining them.

What about more general expressions involving selection bindings?

Key property: each S_w valuation specifies a different channel

Yes: $s + 2$, Yes: $s * 3$

Channel arrays, (some) Selection bindings

$$E = \{(S_1, A_1, g_1, \langle e_1^1, \ldots, e_{n_C}^1 \rangle), \ldots, (S_m, A_m, g_m, \langle e_1^m, \ldots, e_{n_C}^m \rangle)\}$$

e_i — expression over state variables

e_i — single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

chan c[9];

$(\{s : [3,5]\}, A, g, \langle s \rangle)$

$\Longrightarrow (\{s : [0,8]\}, A, g \wedge (s \geq 3) \wedge (s \leq 5), \langle s \rangle)$

- It can be done.
- No detail in this presentation!
- Introduce selection bindings to cover all channels. . .
- . . . add a predicate to each transition before joining them.

What about more general expressions involving selection bindings?

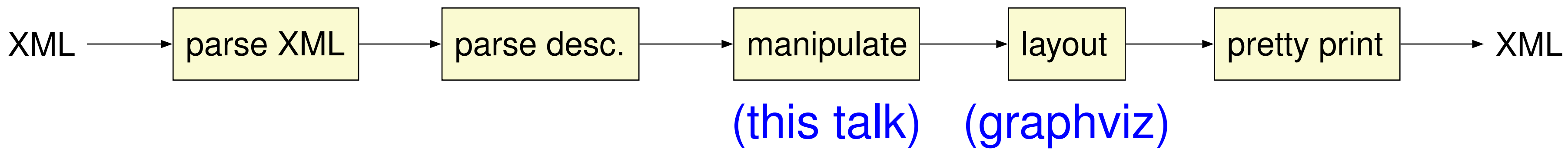
Key property: each S_w valuation specifies a different channel

Yes: $s + 2$, Yes: $s * 3$ No: $s \bmod 5$, No: $(\lambda x.1)\, s$

Presentation Outline

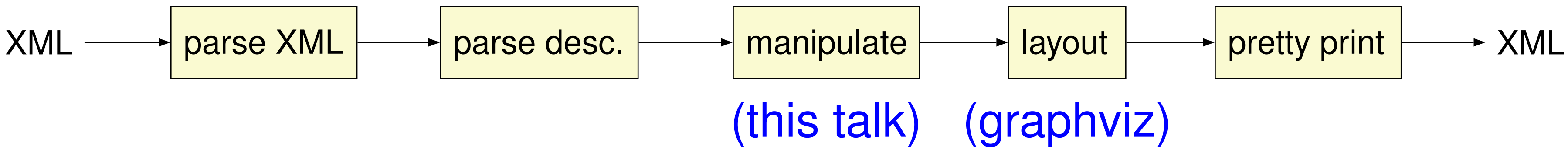
- $\surd$ Testing timed trace inclusion
- $\surd$ Automation and Uppaal features
- $\surd$ Basic guards
- $\surd$ Selection bindings
- $\surd$ Quantifiers
- $\surd$ Channel arrays
- $\Rightarrow$ Implementation
- Summary

Implementation: urpal



- Written in (mostly functional) Standard ML
- Our basic library is generic and BSD-licensed (`libutap` is not required)
- Includes some other manipulations
- Source code and binaries online, google: **urpal**

Implementation: urpal



- Written in (mostly functional) Standard ML
- Our basic library is generic and BSD-licensed (`libutap` is not required)
- Includes some other manipulations
- Source code and binaries online, google: **urpal**

Validating determinism and tool

$$\neg\text{fault} \wedge \text{deterministic}(\mathcal{S}) \implies (\mathcal{S} \parallel \mathcal{S}' \models \text{A}\Box\,\neg\text{Err})$$

- The construction does not depend on determinism
- A precise check must consider the reachable state space

Summary

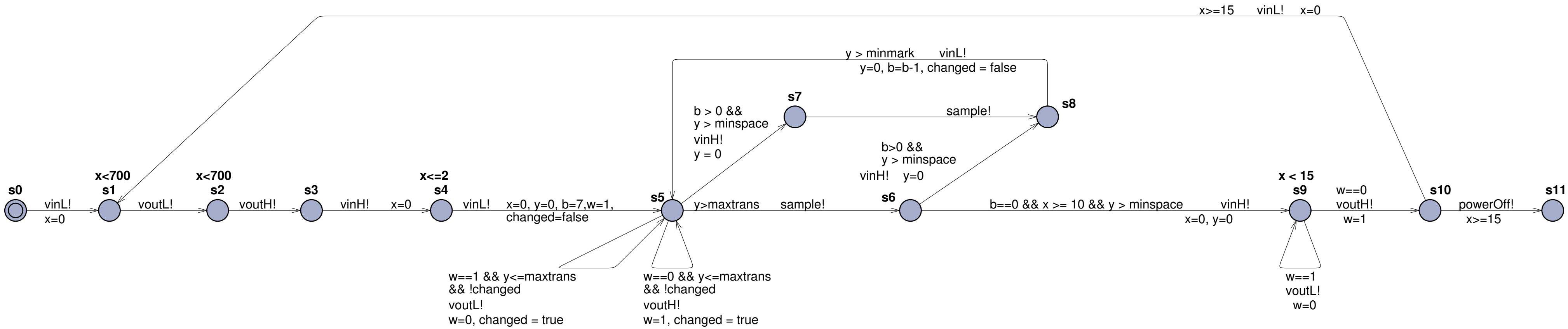
- Introduction to a construction for deciding timed trace inclusion
- Various tricks needed for various features of Uppaal
- Implemented (mostly) and available online

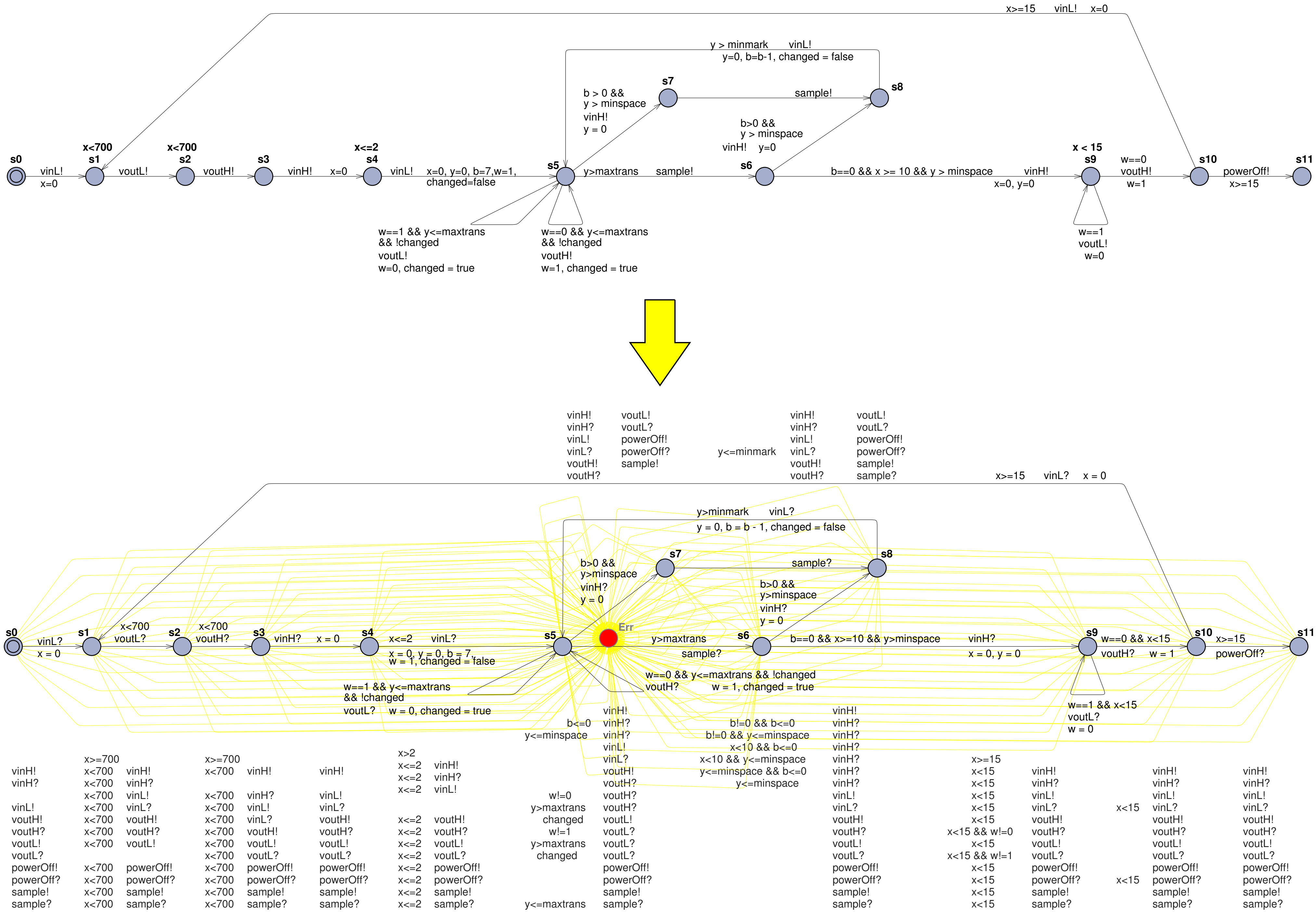
Summary

- Introduction to a construction for deciding timed trace inclusion
- Various tricks needed for various features of Uppaal
- Implemented (mostly) and available online

Further work

- Improve simplification of terms (connect with other tools?)
- Is it easier in Uppaal TIGA?


x>=15 vinL! x=0
y > minmark vinL!
y=0, b=b-1, changed = false
s7
s8
sample!
b > 0 &&
y > minspace
vinH!
y = 0
b>0 &&
y > minspace
vinH! y=0
x<700
x<700
x<=2
x < 15
s0
s1
s2
s3
s4
s5
s6
s9
s10
s11
vinL!
x=0
voutL!
voutH!
vinH!
x=0
vinL!
x=0, y=0, b=7,w=1,
changed=false
y>maxtrans
sample!
b==0 && x >= 10 && y > minspace
vinH!
x=0, y=0
w==0
voutH!
w=1
powerOff!
x>=15
w==1 && y<=maxtrans
&& !changed
voutL!
w=0, changed = true
w==0 && y<=maxtrans
&& !changed
voutH!
w=1, changed = true
w==1
voutL!
w=0


x>=15 vinL! x=0
y > minmark vinL!
y=0, b=b-1, changed = false
s7
s8
sample!
b > 0 &&
y > minspace
vinH!
y = 0
b>0 &&
y > minspace
vinH! y=0
s0
vinL!
x=0
x<700
s1
voutL!
x<700
s2
voutH!
s3
vinH!
x=0
x<=2
s4
vinL!
x=0, y=0, b=7,w=1,
changed=false
s5
y>maxtrans
sample!
s6
b==0 && x >= 10 && y > minspace
vinH!
x=0, y=0
x < 15
s9
w==0
voutH!
w=1
s10
powerOff!
x>=15
s11
w==1 && y<=maxtrans
&& !changed
voutL!
w=0, changed = true
w==0 && y<=maxtrans
&& !changed
voutH!
w=1, changed = true
w==1
voutL!
w=0
x>=15 vinL? x = 0
y>minmark vinL?
y = 0, b = b - 1, changed = false
sample?
b>0 &&
y>minspace
vinH?
y = 0
Err
vinL?
x = 0
x<700
voutL?
voutH?
vinH?
x = 0
x<=2 vinL?
x = 0, y = 0, b = 7,
w = 1, changed = false
y>maxtrans
sample?
b==0 && x>=10 && y>minspace
vinH?
x = 0, y = 0
w==0 && x<15
voutH? w = 1
x>=15
powerOff?
w==0 && y<=maxtrans && !changed
voutH? w = 1, changed = true
w==1 && y<=maxtrans
&& !changed
voutL? w = 0, changed = true
w==1 && x<15
voutL?
w = 0

References

[Alur and Dill, 1994] Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235.

[Henzinger et al., 1992] Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. (1992). Symbolic model checking for real-time systems. In *Proc. 7th Annual IEEE Symposium on on Logic in Computer Science (LICS '92)*, pages 394–406, Santa Cruz, USA. IEEE Computer Society.

[Jensen et al., 2000] Jensen, H. E., Larsen, K. G., and Skou, A. (2000). Scaling up Uppaal: Automatic verification of real-time systems using compositionality and abstraction. In Joseph, M., editor, *Proc. 6th Int. Symposium on Formal Techniques for Real-Time and Fault-Tolerance (FTRTFT '00)*, volume 1926 of *Lecture Notes in Computer Science*, pages 19–30, Pune, India. Springer-Verlag.

[Larsen et al., 1997] Larsen, K. G., Pettersson, P., and Wang, Y. (1997). Uppaal in a nutshell. *Int. Journal of Software Tools for Technology Transfer*, 1(1–2):134–152.

[Stoelinga, 2002] Stoelinga, M. I. (2002). *Alea Jacta est: Verification of probabilistic, real-time and parametric systems*. PhD thesis, Katholieke Universiteit Nijmegen, The Netherlands.