Automatically transforming and relating Uppaal models of embedded systems

Timothy Bourke  CSE UNSW and NICTA
Arcot Sowmya  CSE UNSW
GP2D02 Timing Chart

x >= 15 \ vinL! \ x = 0

0.2 ms or less
1 ms or more
70 ms or more

Power OFF

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

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

Parameter Symbol Rating Unit

Supply voltage Input terminal voltage Output terminal voltage Operating temperature Storage temperature

VCC - 0.3 to +10 V Vin - 0.3 to +3 V
BVO - 0.3 to +10 V T_{opr} -10 to +60 \degree C T_{stg} -40 to +70 \degree C

In the absence of confirmation by device specification sheets, SHARP takes no responsibility for any defects that occur in equipment using any of SHARP's devices, shown in catalogs, data books, etc. Contact SHARP in order to obtain the latest version of the device specification sheets before using any SHARP's device.
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
4. Capable of changing of distance measuring range through change the optical portion (lens)

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

Parameter Symbol Rating Unit
- Supply voltage: VCC 4.4 to 7 V
- Input terminal voltage: Vin - 0.3 to +3 V
- Output terminal voltage: BVO - 0.3 to +10 V
- Operating temperature: Opr -10 to +60 ℃
- Storage temperature: Tr -25 to +85 ℃
- Reflective object:
  - White paper: KODAK made gray chart R-27, white surface (reflectivity: 90%)
  - Gray paper: KODAK made gray chart R-27, gray surface (reflectivity: 50%)

Globally, SHARP Corporation shall not be liable for any defects that occur in equipment using any of SHARP’s devices, shown in catalogs, data books, etc. Contact SHARP in order to obtain the latest version of the device specification sheets before using any SHARP’s device.

VIN EQU P1.0
VOUT EQU P1.1
W100US EQU 50

PREAD: PUSH IE
CLR EA
CLR VIN
NOP
NOP
JB VOUT, RET

LOOP: SETB VIN
MOV R1, #W100US
DJNZ R1, *
CLR VIN
MOV VOUT, C
RLC A
NOP
DJNZ R0, LOOP
JB VOUT, RET
SETB VIN
POP IE
MOV R0, #8
RET
Compact, High Sensitive Distance Measuring Sensor

**Features**
- Impervious to color and reflectivity of reflective object
- High precision distance measurement output for direct connection to microcomputer
- Capable of changing of distance measuring range through change the optical portion (lens)

**Applications**
- Sanitary sensors
- Human body sensors for consumer products such as electric fans and air conditioners
- Garage sensors

**Parameter Symbol**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Rating</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply voltage</td>
<td>VCC</td>
<td>4.4 to 7 V</td>
<td></td>
</tr>
<tr>
<td>Input terminal voltage</td>
<td>Vin</td>
<td>-0.3 to +3 V</td>
<td></td>
</tr>
<tr>
<td>Output terminal voltage</td>
<td>Vout</td>
<td>-0.3 to +10 V</td>
<td></td>
</tr>
<tr>
<td>Operating temperature</td>
<td>Temp</td>
<td>-10 to +60°C</td>
<td></td>
</tr>
<tr>
<td>Storage temperature</td>
<td>Temp</td>
<td>-40 to +70°C</td>
<td></td>
</tr>
</tbody>
</table>

**TIMEDIAG**

**VI**

**DRIVER || SENSOR**

**VI**

**MCS51**

```
VIN EQU P1.0
VOUT EQU P1.1
W100US EQU 50

LOOP: SETB VIN

MOV R1, #W100US
DJNZ R1, *
CLR VIN

PREAD: PUSH IE
CLR EA
CLR VIN
NOP
NOP
DJNZ R0, LOOP
JB VOUT, *
JNB VOUT, *
MOV R0, #8
RET
```
TIMEDIAG

VI

DRIVER || SENSOR

VI

MCS51

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

implements

\mathcal{I} \leq \mathcal{S}

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

must be deterministic

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

**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

**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

$\mathcal{I}_1$: $i_0$ y < 2

y >= 1
a!

y < 3

b!
i_1

$S$: $s_0$
a!
x = 0

x<7

$s_1$

x<5
b!

x>3
c?

$s_2$

$s_3$

Does $\mathcal{I}_1$ implement $S$?
Testing timed trace inclusion

$S$:

$S'$:

- $S'$ is a testing automaton for $S$
- New Err state
- Actions are complemented
- Invariants are shifted
Testing timed trace inclusion

- Run $S'$ in parallel with $I_1$

- Is $\text{Err}$ reachable?
Testing timed trace inclusion

$\mathcal{I}_1$: $y \in [1, 2)$

$s_0$ $x \in [1, 2)$

$s_1$ $x = 0$

$s_2$ $x < 5$

$s_3$ $x > 3 \land x < 7$

Err
Testing timed trace inclusion

\[ \mathcal{I}_1: \quad y \in [1, 2) \]

\[ \mathcal{S}': \quad x \in [1, 2) \]

- \( y - x \in [1, 2) \)
- **Err** is not reachable, therefore \( \mathcal{I}_1 \leq_{ttr} \mathcal{S} \)
Testing timed trace inclusion

\[ \mathcal{I}_2: \]
- \( i_0 \)
  - \( y < 2 \)
  - \( y \geq 1 \)
  - \( a! \)
- \( i_1 \)
  - \( y < 7 \)
- \( i_2 \)

\[ \leq \text{ttr} \]

\[ \mathcal{S}: \]
- \( s_0 \)
  - \( a! \)
  - \( x = 0 \)
- \( s_1 \)
  - \( x < 7 \)
- \( s_2 \)
  - \( x < 5 \)
  - \( b! \)
- \( s_3 \)
  - \( x > 3 \)
  - \( c? \)

- Change the invariant on \( i_1 \), from \( y < 3 \) to \( y < 7 \).
- Does \( \mathcal{I}_2 \) implement \( \mathcal{S} \)?
Testing timed trace inclusion

- Use the same test automaton $S'$
- Is Err reachable?
Testing timed trace inclusion

\[ I_2: \quad y \in [1, 2) \]

\[ S': \quad x \in [1, 2) \]

- So far so good...
Testing timed trace inclusion

$\mathcal{I}_2$: $y \in [1, 2)$

$\mathcal{S}'$: $x \in [1, 2)$

$\mathcal{I}_2$: $y \in [5, 6)$

$\mathcal{S}'$: $x \in [6, 7)$

- $y - x \in [1, 2)$

- $\text{Err}$ is reachable, therefore $\mathcal{I} \not\preceq_{\text{ttr}} \mathcal{S}$, 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]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>urgent nodes</td>
<td>✓</td>
</tr>
<tr>
<td>urgent channels</td>
<td>✓</td>
</tr>
<tr>
<td>shared variables</td>
<td>✓</td>
</tr>
<tr>
<td>selection bindings</td>
<td>✓✓</td>
</tr>
<tr>
<td>quantifiers</td>
<td>✓✓</td>
</tr>
<tr>
<td>channel arrays</td>
<td>✓✓</td>
</tr>
<tr>
<td>committed nodes</td>
<td>X</td>
</tr>
<tr>
<td>broadcast channels</td>
<td></td>
</tr>
<tr>
<td>process priorities</td>
<td></td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>urgent nodes</td>
<td>✔</td>
</tr>
<tr>
<td>urgent channels</td>
<td>✔</td>
</tr>
<tr>
<td>shared variables</td>
<td>✔</td>
</tr>
<tr>
<td>selection bindings</td>
<td>✔ ✔</td>
</tr>
<tr>
<td>quantifiers</td>
<td>✔ ✔</td>
</tr>
<tr>
<td>channel arrays</td>
<td>✔ ✔</td>
</tr>
<tr>
<td>committed nodes</td>
<td>✗</td>
</tr>
<tr>
<td>broadcast channels</td>
<td>✔ ✗</td>
</tr>
<tr>
<td>process priorities</td>
<td></td>
</tr>
</tbody>
</table>
**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]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>urgent nodes</td>
<td>✔️</td>
</tr>
<tr>
<td>urgent channels</td>
<td>✔️</td>
</tr>
<tr>
<td>shared variables</td>
<td>✔️</td>
</tr>
<tr>
<td>selection bindings</td>
<td>✔️ ✔️</td>
</tr>
<tr>
<td>quantifiers</td>
<td>✔️ ✔️</td>
</tr>
<tr>
<td>channel arrays</td>
<td>✔️ ✔️</td>
</tr>
<tr>
<td>committed nodes</td>
<td>❌</td>
</tr>
<tr>
<td>broadcast channels</td>
<td>✔️ ❌</td>
</tr>
<tr>
<td>process priorities</td>
<td>❌</td>
</tr>
</tbody>
</table>
Uppaal transition features

- Basic guards
- Selection bindings
- Universal quantifiers in guards
- Channel arrays
No selection bindings, No quantifiers, No channel arrays

Group by state / channel / direction:

1. Join
2. Negate
3. DNF / Simplify

4. Split

\[
\begin{align*}
&x \geq 1 \land y < 4 \\
&x \leq 3 \land y \geq 2
\end{align*}
\]

\text{clock } x, y;
No selection bindings, No quantifiers, No channel arrays

Group by state / channel / direction:

1. Join

\[(x \geq 1 \land y < 4) \lor (x \leq 3 \land y \geq 2)\]

2. Negate

3. DNF / Simplify

4. Split

```plaintext
clock x, y;
```

```
x \geq 1 \land y < 4
```

```
x \leq 3 \land y \geq 2
```

```
x < 1 \land y < 2
```

```
y \geq 4 \land x > 3
```

```
x \leq 3 \land y \geq 2
```

```
x \geq 1 \land y < 4
```

```
x < 1 \land y < 2
```

```
No selection bindings, No quantifiers, No channel arrays

Group by state / channel / direction:

1. Join
   \[(x \geq 1 \land y < 4) \lor (x \leq 3 \land y \geq 2)\]

2. Negate
   \[(x < 1 \lor y \geq 4) \land (x > 3 \lor y < 2)\]

3. DNF / Simplify

4. Split

\[
\begin{align*}
\text{clock } x, y;
\end{align*}
\]
No selection bindings, No quantifiers, No channel arrays

Group by state / channel / direction:

1. Join
   \[(x \geq 1 \land y < 4) \lor (x \leq 3 \land y \geq 2)\]

2. Negate
   \[(x < 1 \lor y \geq 4) \land (x > 3 \lor y < 2)\]

3. DNF / Simplify
   \[(x < 1 \land x > 3) \lor (y \geq 4 \land x > 3) \lor (x < 1 \land y < 2) \lor (y \geq 4 \land y < 2)\]

4. Split

\[
\begin{array}{c}
s0 & c & ! \\
\hline
s1 & c! & x \geq 1 \land y < 4 \\
& & x \leq 3 \land y \geq 2 \\
\hline
s2 & c! & y \geq 4 \land x > 3 \\
\end{array}
\]

\[
\text{clock } x, y;
\]
No selection bindings, No quantifiers, No channel arrays

Group by state / channel / direction:

1. Join 
   \[(x \geq 1 \land y < 4) \lor (x \leq 3 \land y \geq 2)\]

2. Negate 
   \[(x < 1 \lor y \geq 4) \land (x > 3 \lor y < 2)\]

3. DNF / Simplify 
   \[(x < 1 \land x > 3) \lor (y \geq 4 \land x > 3) \lor (x < 1 \land y < 2) \lor (y > 4 \land y < 2)\]

4. Split

```plaintext
clock x, y;
```
No selection bindings, No quantifiers, No channel arrays

Group by state / channel / direction:

1. Join

   \[(x \geq 1 \land y < 4) \lor (x \leq 3 \land y \geq 2)\]

2. Negate

   \[(x < 1 \lor y \geq 4) \land (x > 3 \lor y < 2)\]

3. DNF / Simplify

   \[(x < 1 \land x > 3) \lor (y \geq 4 \land x > 3) \lor (x < 1 \land y < 2) \lor (y > 4 \land y < 2)\]

4. Split

   clock \(x, y;\)
Group by state / channel / direction:

1. Join
   \[(x \geq 1 \land y < 4) \lor (x \leq 3 \land y \geq 2)\]

2. Negate
   \[(x < 1 \lor y \geq 4) \land (x > 3 \lor y < 2)\]

3. DNF / Simplify
   \[(x < 1 \land x > 3) \lor (y \geq 4 \land x > 3) \lor (x < 1 \land y < 2) \lor (y > 4 \land y < 2)\]

4. Split

\[\text{clock } x, y;\]

\textbf{Key issue: splitting disjunction}
Selection bindings, No quantifiers, No channel arrays

\[ s0 \quad c \quad ! \]

Group by state / channel / direction:

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

\[ \text{i : int[0,n-1]} \]
\[ x[i] \leq i \]
\[ c! \]

\text{clock} \ x[n];
Selection bindings, No quantifiers, No channel arrays

$$s_0 \quad c \quad !$$

Group by state / channel / direction:

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

$$s_0 \quad c \quad !$$

$$i : \text{int}[0,n-1]$$

$$x[i] \leq i$$

$$c!$$

Sometimes it's easy...
Selection bindings, No quantifiers, No channel arrays

$$s_0 \quad c \quad !$$

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**

$$i : \text{int}[0,n-1]$$

$$x[i] \leq i$$

$$c!$$

$$s_0$$

$$x[0] \leq 0$$

$$c!$$

$$x[1] \leq 1$$

$$c!$$

$$\text{clock } x[n];$$
Selection bindings, No quantifiers, No channel arrays

\[ s_0 \quad \text{c} \quad ! \]

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

Sometimes it's easy.

\[ \text{clock } x[n]; \]
Selection bindings, No quantifiers, No channel arrays

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

\[ i : \text{int}[0,n-1] \]
\[ x[i] \leq i \]
\[ c! \]

\[ x[0] \leq 0 \]
\[ c! \]
\[ x[1] \leq 1 \]
\[ c! \]

\[ x[n-1] \leq n-1 \]
\[ c! \]

\textbf{clock} \ x[n];
Selection bindings, No quantifiers, No channel arrays

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

\[ \text{clock } x[n]; \]
Selection bindings, No quantifiers, No channel arrays

\begin{align*}
\text{s0} & \quad c & \quad ! \\
\end{align*}

Group by state / channel / direction:

1. Join \hspace{1cm} \exists i \in \{0, \ldots, n - 1\}. x_i \leq i

2. Negate \hspace{1cm} \forall i \in \{0, \ldots, n - 1\}. x_i > i

3. DNF / Simplify \hspace{1cm} \forall i \in \{0, \ldots, n - 1\}. x_i > i

4. Split

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
   \[(\exists s_{11}, \ldots, s_{1n_1} \cdot g_1) \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m)\]

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
\[
(\exists s_{11}, \ldots, s_{1n_1} \cdot g_1) \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m)
\]

\[
\exists s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot g_1 \lor \cdots \lor g_m
\]

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

\[ (\exists s_{11}, \ldots, s_{1n_1} \cdot g_1) \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m) \]
\[ \exists s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot g_1 \lor \cdots \lor g_m \]

2. Negate

\[ \forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \neg g_1 \land \cdots \land \neg g_m \]

3. DNF / Simplify

4. Split
Selection bindings, No quantifiers, No channel arrays

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

1. Join
\[ (\exists s_{11}, \ldots, s_{1n_1} \cdot g_1) \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m) \]
\[ \exists s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot g_1 \lor \cdots \lor g_m \]

2. Negate
\[ \forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \neg g_1 \lor \cdots \lor \neg g_m \]

3. DNF / Simplify
\[ \forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \bar{g}_1 \lor \cdots \lor \bar{g}_m' \]

4. Split

... in general, another trick is needed.
**Selection bindings, No quantifiers, No channel arrays**

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

1. **Join**
   \[ \exists s_{11}, \ldots, s_{1n_1} \cdot g_1 \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m) \]
   \[ \exists s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot g_1 \lor \cdots \lor g_m \]

2. **Negate**
   \[ \forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \neg g_1 \land \cdots \land \neg g_m \]

3. **DNF / Simplify**
   \[ \forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \overline{g}_1 \lor \cdots \lor \overline{g}_{m'} \]

4. **Split?**
   \[ (\forall S'_1 \cdot \overline{g}_1) \lor \cdots \lor (\forall S'_{m'} \cdot \overline{g}_{m'}) \]
   not always possible!
Selection bindings, No quantifiers, No channel arrays

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

1. Join
   \[
   (\exists s_1, \ldots, s_{1n_1} \cdot g_1) \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m)
   \]
   \[
   \exists s_1, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot g_1 \lor \cdots \lor g_m
   \]

2. Negate
   \[
   \forall s_1, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \neg g_1 \lor \cdots \lor \neg g_m
   \]

3. DNF / Simplify
   \[
   \forall s_1, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \overline{g}_1 \lor \cdots \lor \overline{g}_m'
   \]

4. Split
   \[
   (\forall S_1' \cdot \overline{g}_1) \lor \cdots \lor (\forall S_m' \cdot \overline{g}_m')
   \]
   not always possible!

```
s : IDXT 
x < a[s] && y > b[s] 
c ?

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
   \[ (\exists s_{11}, \ldots, s_{1n_1} \cdot g_1) \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m) \]
   \[ \exists s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot g_1 \lor \cdots \lor g_m \]

2. Negate
   \[ \forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \neg g_1 \land \cdots \land \neg g_m \]

3. DNF / Simplify
   \[ \forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \bar{g}_1 \lor \cdots \lor \bar{g}_m' \]
   \[ (\forall S'_1 \cdot \bar{g}_1) \lor \cdots \lor (\forall S'_m \cdot \bar{g}_m') \]

not always possible!

\[
\begin{align*}
s &: \text{IDXT} & x < a[s] & \& \ y > b[s] & c? \\
\text{clock} & x, \ y; & \text{typedef scalar[N] IDXT;} & \text{int} & a[\text{IDXT}], b[\text{IDXT}]; \\
\end{align*}
\]

\[
\begin{align*}
s &: \text{IDXT} & x < a[s] & \& \ y > b[s] & c! \\
\text{forall} & (s : \text{IDXT}) & x >= a[s] & \| \ y <= b[s] & c! \\
\end{align*}
\]
Selection bindings, No quantifiers, No channel arrays

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

1. Join
\[
(\exists s_{11}, \ldots, s_{1n_1} \cdot g_1) \lor \cdots \lor (\exists s_{m1}, \ldots, s_{mn_m} \cdot g_m)
\]
\[
\exists s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot g_1 \lor \cdots \lor g_m
\]

2. Negate
\[
\forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \neg g_1 \lor \cdots \lor \neg g_m
\]

3. DNF / Simplify
\[
\forall s_{11}, \ldots, s_{1n_1}, \ldots, s_{m1}, \ldots, s_{mn_m} \cdot \bar{g}_1 \lor \cdots \lor \bar{g}_{m'}
\]
\[
(\forall S'_1 \cdot \bar{g}_1) \lor \cdots \lor (\forall S'_{m'} \cdot \bar{g}_{m'})
\]
not always possible!

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

...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

Devise a predicate canswap(x)

Use a looping construction (if no scalars)

(Not yet)

(Also works for simpler case where \( A_i = \))
Selection bindings, Quantifiers, No channel arrays

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

1. Join

\[ (\exists S_1 \forall A_1 \cdot g_1) \lor \cdots \lor (\exists S_m \forall A_m \cdot g_m) \]

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
   \[
   (\exists S_1 \forall A_1. g_1) \lor \cdots \lor (\exists S_m \forall A_m. g_m)
   \]
   \[
   \exists S_1, \ldots, S_m \forall A_1, \ldots, A_m. g_1 \lor \cdots \lor g_m
   \]

2. Negate

3. DNF / Simplify

4. Split

Devise a predicate \( \text{canswap} \) (if noscalars)

Also works for simplercasewhere \( A_i = \) (notyet) even worse!

Usealoopingconstruction
Selection bindings, Quantifiers, No channel arrays

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

1. Join

\[(\exists S_1 \forall A_1 \cdot g_1) \lor \cdots \lor (\exists S_m \forall A_m \cdot g_m)\]

\[ \exists S_1, \ldots, S_m \forall A_1, \ldots, A_m \cdot g_1 \lor \cdots \lor g_m \]

2. Negate

\[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m \cdot \neg g_1 \land \cdots \land \neg g_m \]

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
   \[(\exists S_1 \forall A_1. g_1) \lor \cdots \lor (\exists S_m \forall A_m. g_m)\]
   \[\exists S_1, \ldots, S_m \forall A_1, \ldots, A_m. g_1 \lor \cdots \lor g_m\]

2. Negate
   \[\forall S_1, \ldots, S_m \exists A_1, \ldots, A_m. \neg g_1 \land \cdots \land \neg g_m\]

3. DNF / Simplify
   \[\forall S_1, \ldots, S_m \exists A_1, \ldots, A_m. g_1 \lor \cdots \lor g_m'\]

4. Split
Selection bindings, Quantifiers, No channel arrays

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

1. Join
   \[ (\exists S_1 \forall A_1. g_1) \lor \cdots \lor (\exists S_m \forall A_m. g_m) \]
   \[ \exists S_1, \ldots, S_m \forall A_1, \ldots, A_m. g_1 \lor \cdots \lor g_m \]

2. Negate
   \[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m. \neg g_1 \land \cdots \land \neg g_m \]

3. DNF / Simplify
   \[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m. \overline{g_1} \lor \cdots \lor \overline{g_m'} \]

4. Split
Selection bindings, Quantifiers, No channel arrays

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

1. Join

\[ (\exists S_1 \forall A_1 . g_1) \lor \cdots \lor (\exists S_m \forall A_m . g_m) \]

\[ \exists S_1, \ldots, S_m \forall A_1, \ldots, A_m . g_1 \lor \cdots \lor g_m \]

2. Negate

\[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m . \neg g_1 \land \cdots \land \neg g_m \]

3. DNF / Simplify

\[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m . \bar{g}_1 \lor \cdots \lor \bar{g}_m' \]

4. Split?

\[ \exists A_1, \ldots, A_m . (\forall S'_1 \cdot \bar{g}_1) \lor \cdots \lor (\forall S'_m \cdot \bar{g}_m') \]

even worse!
Selection bindings, Quantifiers, No channel arrays

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

1. Join
   \[ (\exists S_1 \forall A_1. g_1) \lor \cdots \lor (\exists S_m \forall A_m. g_m) \]
   \[ \exists S_1, \ldots, S_m \forall A_1, \ldots, A_m. g_1 \lor \cdots \lor g_m \]

2. Negate
   \[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m. \neg g_1 \lor \cdots \lor \neg g_m \]

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

4. Split?
   \[ \exists A_1, \ldots, A_m. (\forall S'_1. \overline{g}_1) \lor \cdots \lor (\forall S'_{m'}. \overline{g}_{m'}) \]
   even worse!

- Devise a predicate \(\text{canswap}(\varphi)\)
- 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
\[ (\exists S_1 \forall A_1 . g_1) \lor \cdots \lor (\exists S_m \forall A_m . g_m) \]
\[ \exists S_1, \ldots, S_m \forall A_1, \ldots, A_m . g_1 \lor \cdots \lor g_m \]

2. Negate
\[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m . \neg g_1 \land \cdots \land \neg g_m \]

3. DNF / Simplify
\[ \forall S_1, \ldots, S_m \exists A_1, \ldots, A_m . g_1 \lor \cdots \lor \neg g_m' \]

4. Split
\[ \exists A_1, \ldots, A_m . (\forall S'_1 . \neg g_1) \lor \cdots \lor (\forall S'_m . \neg g_m') \]

- Devise a predicate \(\text{canswap}(\varphi)\)

- Use a looping construction (if no scalars)

(Also works for simpler case where \(A_i = \emptyset\))
Selection bindings, Quantifiers, No channel arrays

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

\[
s_1 : \text{int} [0, N] \\
\text{forall}(a_1 : \text{int} [0, N]) \quad k[s_1] > 3 \land b[s_1][a_1]
\]

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

\[
s_2 : \text{int} [0, N] \\
k[s_2] < 1
\]

\[
\text{bool} \ b[N+1][N+1]; \\
\text{clock} \ k[N+1]; \\
\text{meta int}[0,N] \ t1, \ t2;
\]
Selection bindings, Quantifiers, No channel arrays

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

\[
s_1 : \text{int} \ [0, N] \\
\text{forall}(a_1 : \text{int} \ [0, N]) \quad k[s_1] > 3 \ \&\& \ b[s_1][a_1]
\]

\[
s_2 : \text{int} \ [0, N] \\
k[s_2] < 1
\]

\[
\text{bool} \ b[N+1][N+1]; \\
\text{clock} \ k[N+1];
\]
Selection bindings, Quantifiers, No channel arrays

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

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

\[ s_1 : \text{int} [0, N] \]

\[ \forall a_1 : \text{int} [0, N] \quad k[s_1] > 3 \land b[s_1][a_1] \]

\[ s_2 : \text{int} [0, N] \]

\[ k[s_2] < 1 \]

\[ t_1 = 0, \ t_2 = 0 \]

\[ \text{bool} \ b[N+1][N+1]; \]

\[ \text{clock} \ k[N+1]; \]

\[ \text{meta int}[0,N] \ t1, \ t2; \]
Selection bindings, Quantifiers, No channel arrays

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

\( s_1 : \text{int} [0, N] \)
\( \forall (a_1 : \text{int} [0, N]) \ k[s_1] > 3 \land b[s_1][a_1] \)

\( t_1 = 0, t_2 = 0 \)

\( s_2 : \text{int} [0, N] \)
\( k[s_2] < 1 \)

\[ \text{bool} \ b[N+1][N+1]; \]
\[ \text{clock} \ k[N+1]; \]
\[ \text{meta} \ \text{int}[0,N] \ t_1, t_2; \]
Selection bindings, Quantifiers, No channel arrays

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

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

s_1 \colon\ \text{int } [0, N]
forall(a_1 :\ \text{int } [0, N])\ \ k[s_1] > 3 \land b[s_1][a_1]

forall(a_1 :\ \text{int } [0, N])\ \ k[t_1] > 3 \land b[t_1][a_1]

k[t_2] < 1

\begin{align*}
t_1 &= 0,\ t_2 = 0 \\
t_1 < N \land k[t_1] \leq 3 \land k[t_2] \geq 1 \\
t_1 &= t_1 + 1
\end{align*}

s_2 \colon\ \text{int } [0, N]

k[s_2] < 1

a_1 \colon\ \text{int } [0, N]
\begin{align*}
t_1 < N \land \neg b[t_1][a_1] \land k[t_2] \geq 1 \\
t_1 &= t_1 + 1
\end{align*}

bool b[N+1][N+1];
clock k[N+1];
meta int[0,N] t1, t2;
Selection bindings, Quantifiers, No channel arrays

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

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

$s_1 : \text{int } [0, N]$

forall $a_1 : \text{int } [0, N]$ \quad k[s_1] > 3 \&\& b[s_1][a_1]$

forall $a_1 : \text{int } [0, N]$ \quad k[t_1] > 3 \&\& b[t_1][a_1]$

$k[t_2] < 1$

t_1 = N \&\& t_2 < N \&\& k[t_1] \leq 3 \&\& k[t_2] \geq 1$

t_1 = 0, \ t_2 = t_2 + 1$

t_1 < N \&\& k[t_1] \leq 3 \&\& k[t_2] \geq 1$

t_1 = t_1 + 1$

$t_1 = 0, \ t_2 = 0$

$s_2 : \text{int } [0, N]$

$k[s_2] < 1$

$a_1 : \text{int } [0, N]$

$t_1 < N \&\& \neg b[t_1][a_1] \&\& k[t_2] \geq 1$

t_1 = t_1 + 1$

bool $b[N+1][N+1]$;
clock $k[N+1]$;
meta int[0,N] $t_1$, $t_2$;
Selection bindings, Quantifiers, No channel arrays

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

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

$s_1 : \text{int} [0, N]$
forall$(a_1 : \text{int} [0, N]) \ k[s_1] > 3 \land b[s_1][a_1]$

$s_2 : \text{int} [0, N]$
$k[s_2] < 1$

$t_1 = 0, \ t_2 = 0$

forall$(a_1 : \text{int} [0, N])$
$k[t_1] > 3 \land b[t_1][a_1]$

$k[t_2] < 1$
$t_1 = 0, \ t_2 = t_2 + 1$

$t_1 < N \land k[t_1] \leq 3 \land k[t_2] \geq 1$
$t_1 = t_1 + 1$

$t_1 = N \land t_2 < N \land k[t_1] \leq 3 \land k[t_2] \geq 1$
$t_1 = 0, \ t_2 = t_2 + 1$

bool $b[N+1][N+1]$;
clock $k[N+1]$;
meta int[0,N]$ t1, t2;
Selection bindings, Quantifiers, No channel arrays

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

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

s_1 : \text{int} [0, N]

\text{forall}(a_1 : \text{int} [0, N]) \ k[s_1] > 3 \ & \ b[s_1][a_1]

Conjecture that this always works (for bounded integers)
Channel arrays, No Selection bindings

- Channel array notation: `chan c[N][ST];`
- Bound integers and scalars
- Multidimensional channels

Two possible groupings:

1. $e_1 = e_2$,
   - $g_1$ and $g_2$ cover other channels
2. $e_1 = \neg e_2$,
   - $\neg g_1$ and $\neg g_2$ cover other channels
Channel arrays, No Selection bindings

Group by state / channel / direction

```plaintext
chan c[N][ST];
```

bounded integers
scalars
multidimensional
Channel arrays, No Selection bindings

\[ \text{chan } c[N][ST]; \]

s0 !

Group by state / channel / direction

\( c \) is a set of channels

bounded integers
scalars

multidimensional
Channel arrays, No Selection bindings

\[ \text{s0} \]

Group by state /channel/ direction

\[ \text{c is a set of channels} \]

\[ \text{chan } \text{c}[N][ST]; \]

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

Group by state / channel / direction

c is a set of channels

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

\[ e.g. c \]

\[ e.g. [2 \times i][s][3] \]
Channel arrays, No Selection bindings

Group by state /channel/ direction

c is a set of channels

Example:

```
c
```

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

Two possible groupings:

- \( e_1 = e_2 \)  
  - negate \( g_1 \lor g_2 \)
  - cover other channels

- \( e_1 \neq e_2 \)  
  - negate \( g_1 \)
  - negate \( g_2 \)
  - cover other channels
Channel arrays, (some) Selection bindings

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

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 \( S \) valuation species a different channel

Yes:

- \( s + 2 \)
- \( s \mod 5 \)

No:

- \((x:1) s\)
Channel arrays, (some) Selection bindings

\[ E = \{ (S_1, A_1, g_1, \langle e_{11}^1, \ldots, e_{n_C}^1 \rangle), \ldots, (S_m, A_m, g_m, \langle e_{11}^m, \ldots, e_{n_C}^m \rangle) \} \]

- expression over state variables
- single selection binding over whole range
- only possibility for scalar types
- but integer bindings may span subintervals

```c
chan c[9];

({s : [3, 5]}, A, g, \langle s \rangle)
```
Channel arrays, (some) Selection bindings

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

expression over state variables

single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

\textbf{chan} c[9];

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

\( \implies (\{ s : [0, 8] \}, A, g \land (s \geq 3) \land (s \leq 5), \langle s \rangle) \)
Channel arrays, (some) Selection bindings

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

- It can be done.

Expression over state variables

Single selection binding over whole range

Only possibility for scalar types

But integer bindings may span subintervals

```plaintext
chan c[9];

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

\implies (\{s : [0, 8]\}, A, g \land (s \geq 3) \land (s \leq 5), \langle s \rangle)
```
Channel arrays, (some) Selection bindings

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

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

expression over state variables

single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

\textbf{chan} c[9];

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

\(\Rightarrow (\{ s : [0, 8] \}, A, g \land (s \geq 3) \land (s \leq 5), \langle s \rangle)\)
Channel arrays, (some) Selection bindings

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

- 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.

single selection binding over whole range

expression over state variables

single selection binding over whole range

only possibility for scalar types

but integer bindings may span subintervals

\texttt{chan c[9];}

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

\(\Rightarrow (\{s : [0, 8]\}, A, g \land (s \geq 3) \land (s \leq 5), \langle s \rangle)\)
Channel arrays, (some) Selection bindings

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

- 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

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

- 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: \( s \times 3 \)
Channel arrays, (some) Selection bindings

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

- 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.

expression over state variables
single selection binding over whole range
only possibility for scalar types
but integer bindings may span subintervals

\begin{align*}
\text{chan } & c[9]; \\
& (\{s : [3, 5]\}, A, g, \langle s \rangle) \\
\implies & (\{s : [0, 8]\}, A, g \land (s \geq 3) \land (s \leq 5), \langle s \rangle)
\end{align*}

What about more general expressions involving selection bindings?

Key property: each \( S_w \) valuation specifies a different channel

Yes: \( s + 2 \), \ Yes: \( s \times 3 \) \  No: \( s \mod 5 \), \ No: \( (\lambda x.1) \ s \)
Presentation Outline

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

```latex
\begin{align*}
\text{XML} & \xrightarrow{\text{parse XML}} \text{parse desc.} \xrightarrow{} \text{manipulate} \xrightarrow{} \text{layout} \xrightarrow{} \text{pretty print} \rightarrow \text{XML} \\
\text{(this talk)} & \quad \text{(graphviz)}
\end{align*}
```
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

\[ \neg \text{fault} \land \text{deterministic}(S) \implies (S \parallel S' \models A\Box \neg \text{Err}) \]

- 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?
References


