Precise Deadlock Detection for Polychronous Data-flow Specifications

V.C. Ngo    J-P. Talpin    T. Gautier

INRIA, Rennes

ESLsyn - DAC 2014
Outline

1. Signal language
2. Problem statement
3. A more precise deadlock detection
4. Implementation with SMT
5. Concluding remarks
Signal and Clock

- **signal** \( x \): sequences \( x(t), t \in \mathbb{N} \), of typed values (\# is absence)
- **clock** \( C_x \) of \( x \): instants where \( x(t) \neq \# \)
- **process**: relations between values/clocks of signals
- **parallelism**: processes are running concurrently
- **process** \( y := x + 1, \forall t \in C_y, y(t) = x(t) + 1 \)

<table>
<thead>
<tr>
<th>( t )</th>
<th>( t_0 )</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1</td>
<td>3</td>
<td>#</td>
<td>6</td>
<td>#</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>( C_x )</td>
<td>tt</td>
<td>tt</td>
<td>ff</td>
<td>tt</td>
<td>ff</td>
<td>tt</td>
<td>...</td>
</tr>
<tr>
<td>( y )</td>
<td>2</td>
<td>4</td>
<td>#</td>
<td>7</td>
<td>#</td>
<td>3</td>
<td>...</td>
</tr>
</tbody>
</table>

- Other languages: Esterel (Esterel Technologies), Lustre (Verimag),...
Primitive Operators

- **Stepwise functions**: \( y := f(x_1, ..., x_n) \)
  \( \forall t \in C_y, y(t) = f(x_1(t), ..., x_n(t)), C_y = C_{x_1} = ... = C_{x_n} \)

- **Delay**: \( y := x^1 \) init \( a \)
  \( y(0) = a, \forall t \in C_x \land t > 0, y(t) = x(t - 1), C_y = C_x \)

- **Merge**: \( y := x \) default \( z \)
  \( y(t) = x(t) \) if \( t \in C_x, y(t) = z(t) \) if \( t \in C_z \setminus C_x, C_y = C_x \cup C_z \)

- **Sampling**: \( y := x \) when \( b \)
  \( \forall t \in C_x \cap C_b \land b(t) = true, y(t) = x(t), C_y = C_x \cap [b] \)

- **Composition**: \( P_1 \| P_2 \) denotes the parallel composition of two processes

- **Restriction**: \( P \) where \( x \) specifies \( x \) as a local variable to \( P \)
Cyclic Dependency Program

```
process CycleDependency =
(?
integer x, c; ! integer v) /* IO signals*/
 |
y := (v when (c <= 0)) default x
| u := y + x /*equations*/
| v := u when (c >= 1) /*order does not matter*/
|
where integer y, u end; /*local signals*/
```

<table>
<thead>
<tr>
<th>t</th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
<td>3</td>
<td>#</td>
<td>2</td>
<td>#</td>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>y</td>
<td>1</td>
<td>3</td>
<td>#</td>
<td>2</td>
<td>#</td>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>u</td>
<td>2</td>
<td>6</td>
<td>#</td>
<td>4</td>
<td>#</td>
<td>14</td>
<td>...</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>6</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>14</td>
<td>...</td>
</tr>
</tbody>
</table>

Ngo, Talpin and Gautier
Precise Deadlock Detection for Polychronous Data-flow Specifications
ESLsyn - DAC 2014
Compilation Process

- Syntax and type analysis
- Clock analysis
- Data dependency analysis
- Code generation

Signal Program → C/C++, Java
Deadlock

- A **deadlock** is a cyclic data dependency, denoted by \((x_0, x_1, ..., x_n, x_0)\)
- In Signal, the dependencies are conditioned by polynomials over \(\mathbb{Z}/3\mathbb{Z}\) that are represented as a **Graph of Conditional Dependency (GCD)**
- This representation may cause **erroneous detection** when dealing with numerical expressions
Example of Deadlock

process CycleDependency =
  (?integer x, c; !integer v)
  (| y := (v when (c <= 0))
      default x
  | u := y + x
  | v := u when (c >= 1)
  )
where integer y, u end;

- $c_1 := c <= 0$, $c_2 := c >= 1$, and $v_1 := v$ when $c_1$
- $x \xrightarrow{P} y$: $y$ depends on $x$ when $\text{Sol}(P - 1) \neq \emptyset$
- The cyclic dependency $(v, v_1, y, u, v)$ stands for a deadlock iff:
  \[
  v^2(-c_1 - c_1^2) \cdot v_1^2 \cdot u^2 \cdot u^2(-c_2 - c_2^2) = 1
  \]
  has some solution

Ngo, Talpin and Gautier
Precise Deadlock Detection for Polychronous Data-flow Specifications
ESLsyn - DAC 2014
Fault Detection

- With current implementation, Signal considers \((v, v_1, y, u, v)\) is a deadlock since
  \[
  \text{Sol}(u^2(-c_2 - c_2^2) * v^2(-c_1 - c_1^2) * v_1^2 * u^2 - 1) \neq 0
  \]
- A solution is \((u^2 = v^2 = v_1^2 = c_1 = c_2 = 1)\), meaning \(c_1\) and \(c_2\) have the value \textit{true} at the same instant
  \(\implies\) Numerical expressions not fully addressed in abstraction: \(c_1\) and \(c_2\) cannot be present at the same instant.
A More Precise Deadlock Detection

- Represent the dependencies as a **Synchronous Data-flow Dependency Graph (SDDG)**
- The dependencies are conditioned by **first-order logic formulas**, called **clock constraints**
- For each signal $x$, attach a pair $(\hat{x}, \tilde{x})$ to encode its clock and value
- Given variation intervals of input signals, the encoding scheme identifies the variation intervals of output and local signals
  - $\phi(b := b_1 \text{ and } b_2) = \tilde{b} = \tilde{b}_1 \land \tilde{b}_2$
  - $\phi(e := c <= 0) = \tilde{e} \iff (\tilde{c} \in (-\infty, 0])$
The clock constraints:

\[ (\hat{v} \Leftrightarrow \hat{u} \land \hat{c}_2 \land \hat{\bar{c}}_2); (\hat{\bar{c}}_2 \Leftrightarrow (\hat{\bar{c}} \in [1, +\infty)))\]

\[ (\hat{v}_1 \Leftrightarrow \hat{v} \land \hat{c}_1 \land \hat{\bar{c}}_1); (\hat{\bar{c}}_1 \Leftrightarrow (\hat{\bar{c}} \in (-\infty, 0])) \]
The cyclic dependency \((v, v_1, y, u, v)\) is a deadlock iff

\[ M \models (\hat{v}_1 \land \hat{v}_1 \land \hat{u} \land \hat{v}) \]
Replacing the definitions of $\hat{v}$ and $\hat{v}_1$, the cyclic dependency $(v, v_1, y, u, v)$ is not deadlock since

$$M \not\models (\hat{v}_1 \land \hat{v} \land \hat{u} \land \hat{v})$$

since $(\tilde{c}_2 \iff (\tilde{c} \in [1, +\infty))) \land (\tilde{c}_1 \iff (\tilde{c} \in (-\infty, 0])) \iff false$

More precise than the current deadlock detection when dealing with numerical expressions, specially the numerical comparisons.
Implementation with SMT

Implementation

Signal Program

Interval Analyzer

Boolean-interval Abstraction

SDDG Construction and Proven Formulas

SMT Solver

1

2

3

4
Related Works

- Gamatié et al. "Enhancing the Compilation of Synchronous Data-flow Programs with Combined Numerical-Boolean Abstraction", 2012
- Jose et al. "SMT based false causal loop detection during code synthesis from polychronous specifications", 2011
- Ngo et al. "Formal Verification of Synchronous Data-flow Program Transformations Toward Certified Compilers", 2013
Concluding Remarks

- An expressive representation of dependency with the Boolean-interval abstraction
- Improvement of static analysis for detecting cyclic dependencies
- Next step: benchmarks and integration in Polychronous toolset
Thanks!

In this talk...

- Signal language
- Problem statement
- A more precise deadlock detection
- Implementation with SMT
- Concluding remarks