Synthesis of Concurrency Constructs using Uninterpreted Functions

Work by: Georg Hofferek, Bettina Koenighofer, Roland Jiang, Ashutosh Gupta
1. Problem
Concurrent: RSA in Software

RSA decryption: Compute

\[ c^d \mod pq = \text{crt}(c^d \mod p, c^d \mod q) \]

**T1::**
1. \( m_p = c^d \mod p \)
2. \( \text{finished}_1 = \text{true} \)
3. if !\( \text{merged} \) && \( \text{finished}_2 \)
4. \( \text{merged} = \text{true} \)
5. \( m_p = \text{crt}(m_p, m_q) \)

**T2::**
1. \( m_q = c^d \mod q \)
2. \( \text{finished}_2 = \text{true} \)
3. if !\( \text{merged} \) && \( \text{finished}_1 \)
4. \( \text{merged} = \text{true} \)
5. \( m_p = \text{crt}(m_p, m_q) \)
Concurency: RSA in Software

Hard to program, easy to specify:

\[ T_1 || T_2 = T_1; T_2 \cup T_2; T_1 \]

**T1::**

1. \( m_p = c^d \mod p \)
2. \( \text{finished}_1 = \text{true} \)
3. if !\text{merged} && \text{finished}_2
4. \( \text{merged} = \text{true} \)
5. \( m_p = \text{crt}(m_p, m_q) \)

**T2::**

1. \( m_q = c^d \mod q \)
2. \( \text{finished}_2 = \text{true} \)
3. if !\text{merged} && \text{finished}_1
4. \( \text{merged} = \text{true} \)
5. \( m_p = \text{crt}(m_p, m_q) \)

Hard to Program, easy to specify

Cannot reason at bit level
Concurrency: RSA in Software

Complex system:
- 64-bit datapath
- very complex arithmetic logic unit
Concurrenty: Pipelined Processor

That's trivial!
Concurrency: Pipelined Processor

IF
r1 := mem[1]
r2 := r1 + r2

DE

MEM
mem[1] = 15

REG
r1 = 15
r2 = 27

EX

ALU
r2 := r1 + r2
r2 := 15 + 2
r2 := 17

MEM

WB
r1 := 15
r2 := 17

stall

forward

15
Concurrency: Pipelined Processor

Not so trivial!
Concurrency: Pipelined Processor

Specification

Hard to Program, easy to specify

ISA

Cannot reason at bit level

Pipeline
### Easy & Hard

<table>
<thead>
<tr>
<th></th>
<th>Specification</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential, data</td>
<td>hard</td>
<td>easy</td>
</tr>
<tr>
<td>parallel</td>
<td>easy</td>
<td>hard</td>
</tr>
</tbody>
</table>

**perfect application area for synthesis:**

**construct synchronization automatically**

Cf. ClarkeEmerson82; VechevYahavYorsh10,…

**Focus today:**

**Uninterpreted functions for complex systems**
2. A Solution
Uninterpreted Functions

Thread1()
1. \( m_1 = c^d \mod p \)
2. \( \text{finished}_1 = \text{true} \)
3. IF !merged AND finished_2
4. merged = true
5. \( m_1 = \text{crt}(m_1, m_2) \)

Thread2()
1. \( m_2 = c^d \mod q \)
2. \( \text{finished}_2 = \text{true} \)
3. IF !merged AND finished_1
4. merged = true
5. \( m_1 = \text{crt}(m_1, m_2) \)

Spec: \( T1 \parallel T2 = T1; T2 \cup T2; T1 \)

Correctness is independent of semantic of functions

Use uninterpreted functions
Concurrency: Pipelined Processor

Specification

ISA

Pipeline
Synthesis = Solving a Quantified Formula

**Lock Synthesis**

\[ \psi = \exists \bar{a}. \forall \text{in}, \text{out}. \Phi \]

- \( \bar{a} \) fixes atomic sections
- \( \bar{a} \): Boolean
- \( \text{in}, \text{out} \): uninterpreted domain

**Pipeline Controller**

\[ \psi = \forall \text{mem}, \text{reg}. \exists \text{stall, fwd}. \forall \text{mem}', \text{reg}'. \Phi \]

- \( \text{stall, fwd} \) fix concurrency
- \( \text{stall, fwd} \): Boolean
- \( \text{mem}, \text{reg} \): uninterpreted domain

\( \Phi \) in Quantifier Free FOL with uninterpreted functions
Clearly decidable
Problem: Efficiently find certificates
Insert Real Work here

Options:
1. Reduce to propositional logic
2. Expand and use interpolation
3. Build a “QBF MT” solver
4. ?
What we can do

Simple processors
Lock synthesis for programs without loops

And what we cannot do

- Temporal aspects, loops
- Combinations with other abstraction techniques?
- More examples… Other logics?
### Ras’ Ten Commandments

<table>
<thead>
<tr>
<th>Commandment</th>
<th>✔️</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Extend the Language</td>
<td>✔️</td>
</tr>
<tr>
<td>2. Don’t reinvent the wheel</td>
<td>✔️</td>
</tr>
<tr>
<td>3. Just Say It</td>
<td>✔️</td>
</tr>
<tr>
<td>4. Don’t assume knowledge of spec language</td>
<td>✔️</td>
</tr>
<tr>
<td>5. Expect the Unexpected</td>
<td>✗</td>
</tr>
<tr>
<td>6. Embrace Ambiguity</td>
<td>✔️</td>
</tr>
<tr>
<td>7. Interact</td>
<td>✗</td>
</tr>
<tr>
<td>8. Invent a Calculus</td>
<td>✗</td>
</tr>
<tr>
<td>9. Divide and conquer</td>
<td>✔️</td>
</tr>
<tr>
<td>10. Talk to Ras</td>
<td>✗</td>
</tr>
</tbody>
</table>