Formal Verification of Compiler Transformations on Polychronous Equations

Van Chan Ngo

Project-Team ESPRESSO

April 12, 2012 – 68nqrt seminar-INRIA/IRISA

work published at IFM 2012

Synchronous data-flow languages

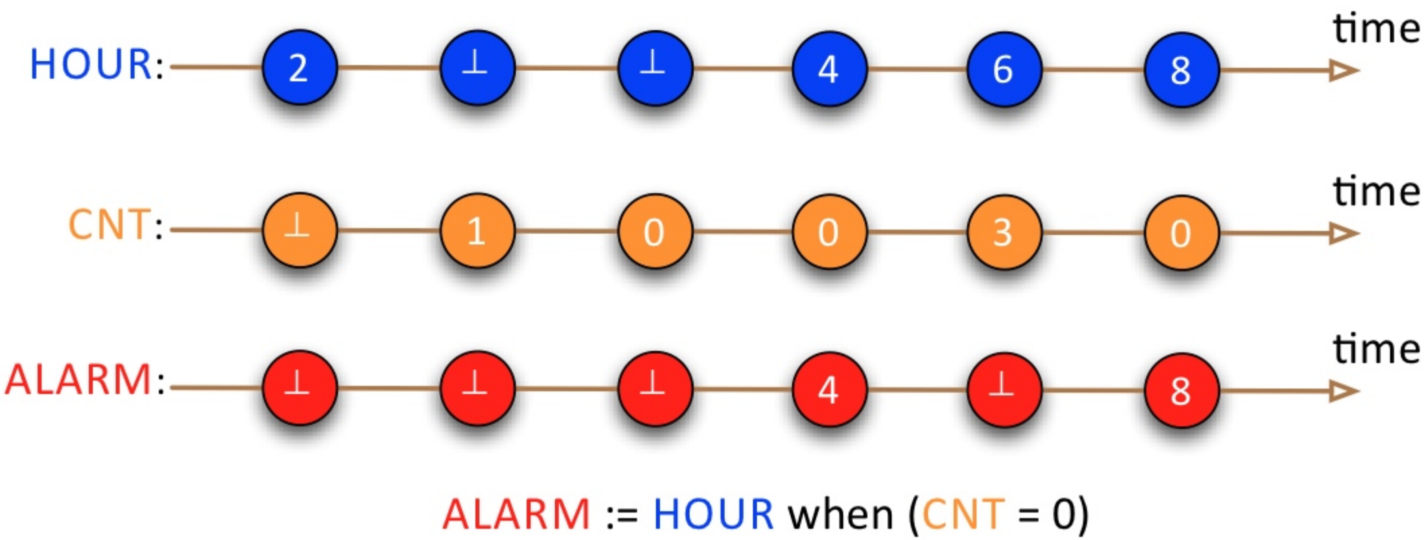
- Have been successfully used in development of embedded and critical real-time systems
- Provide some powerful facilities: simulation, verification, synthesis, code generation,…
- Fulfill the high requirements of efficient and reliable implementations with (at source level): static analysis, model checking, program proof

Polychronous model

- Inputs, outputs: flows of values along time
- Time: discrete and instants are numbered by integers
- Abstract clock: the set of instances that the values of the corresponding data-flow are present
- Flow interactions: specified using clock relations
- System: defined by a system of equations

Examples: Clock Constraint Specification Language, LUSTRE, SIGNAL

An example

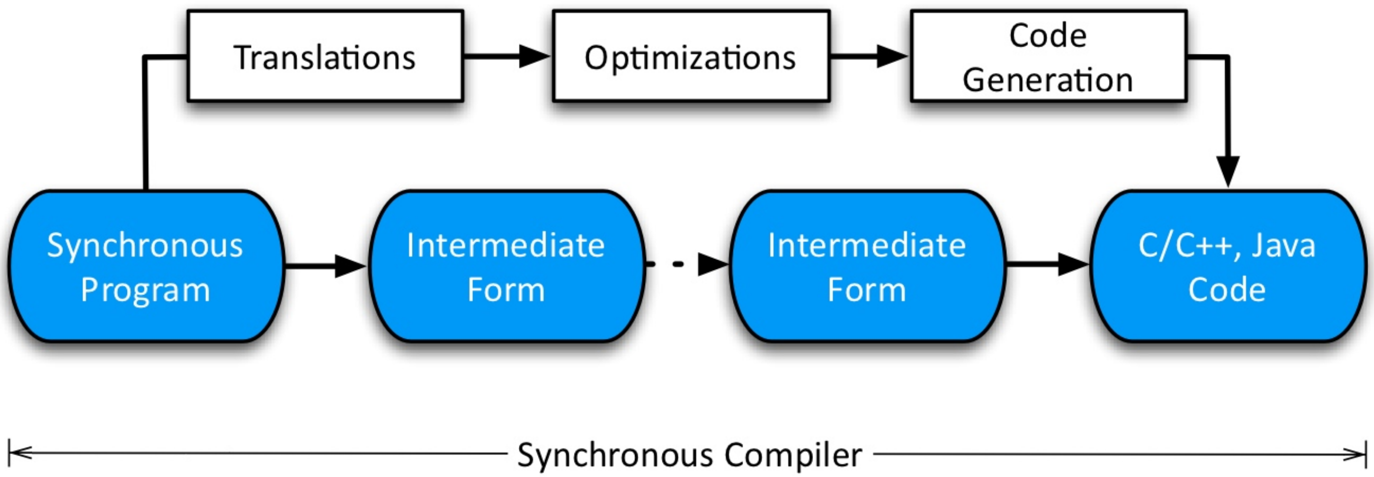


⊥ : Value is absent

Value of ALARM is present if values of HOUR and CNT are present and the value of CNT is 0

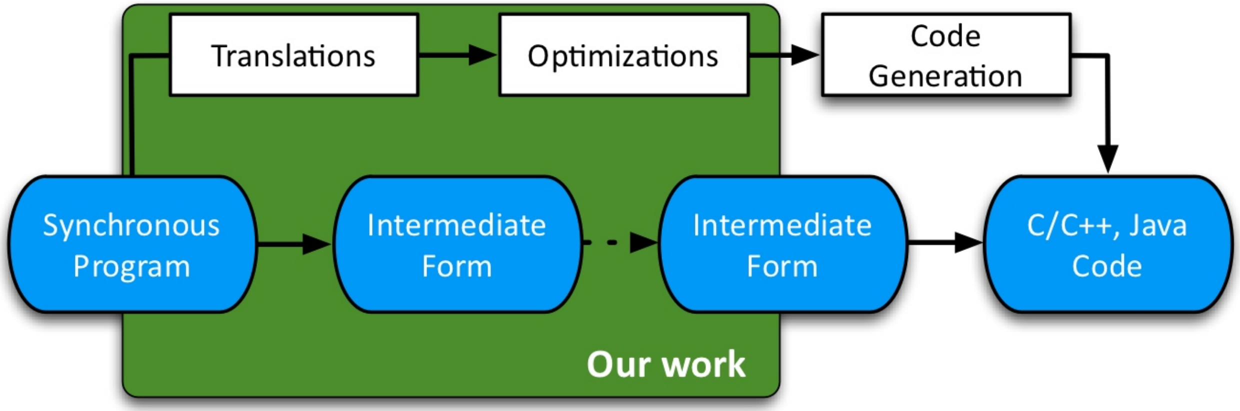
Synchronous compilers

Compiler perform translations, optimizations before generating code



Is there any bug of this compiler?

Objectives



- Prove the correctness of the compiler transformations
- Correctness: ensuring that the abstract clock relation semantics are preserved during the transformations
- An automated process to carry out the proof of the correctness

Approach

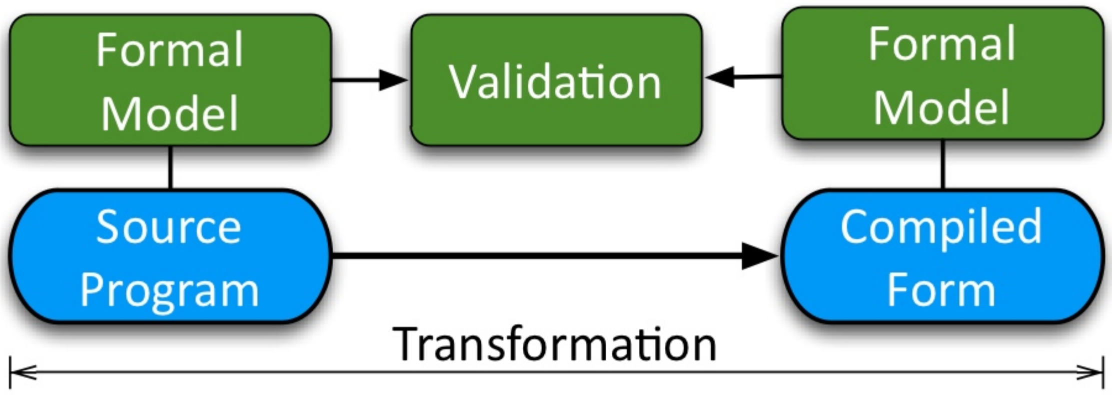
Adopting the translation validation approach of *Pnueli et al*[*], our verification process consists of:

- Formal models capture the abstract clock relation semantics of the source program and its compiled form
- Formal definition of correct transformation (refinement)
- An automated proof method based on simulation techniques

[(*)] *A. Pnueli, O. Shtrichman, and M. Siegel, Translation validation: From SIGNAL to C*

Approach

Each individual transformation is followed by our verification



Advantages

- Avoid the disadvantage of compiler verification approach*
- Independence of how the compiler generates the output from the input
- The verification process is fully automated

(*)Proves the correctness of the compiler in advance then every change requires redoing the proof

Outline

1. The formal model
2. Correct transformation: Refinement
3. Proving refinement by simulation
4. Implementation with SIGALI
5. Conclusion

Formal model

Use a variable x over a finite field modulo 3 to encode the value and status of the data-flow x

Boolean	x	Non-Boolean	x
present $\wedge$ false	-1	present	± 1
absent	0	absent	0
present $\wedge$ true	1		

Then the abstract clock is encoded by x^2

Polynomial Dynamical System

$y = R(x_1,\ldots,x_n)$ of data-flows are represented as a polynomial equation

The program can be modeled as a PDS:

$$\begin{cases} Q(X,Y) & = & 0 \\ X' & = & P(X,Y) \\ Q_0(X) & = & 0 \end{cases}$$

X: state variables (encode the delay operators),

Y: event variables,

X' = P(X,Y): evolution equations,

Q(X,Y) = 0: constraint equations,

$Q_0(X) = 0$: initialization equations.

PDS model of SIGNAL

Boolean signals	
$y := \text{not } x$	$y = -x$
$z := x \text{ and } y$	$z = xy(xy - x - y - 1)$ $x^2 = y^2$
$z := x \text{ or } y$	$z = xy(1 - x - y - xy)$ $x^2 = y^2$
$z := x \text{ default } y$ $z := x \text{ when } y$	$z = x + (1 - x^2)y$ $z = x(-y - y^2)y$
$y := x\$1 \text{ init } y_0$	$\xi' = x + (1 - x^2)\xi$ $y = x^2\xi$ $\xi_0 = y_0$

Non-boolean signals	
$y := f(x_1, ..., x_n)$	$y^2 = x_1^2 = ... = x_n^2$
$z := x \text{ default } y$ $z := x \text{ when } y$	$z^2 = x^2 + y^2 - x^2y^2$ $z^2 = x^2(-y - y^2)$
$y := x\$1 \text{ init } y_0$	$y^2 = x^2$

An example

```
process altern =
  ( ? event A, B;
    ! )
  (| X := not ZX
   | ZX := X$ 1
   | A ^= when X
   | B ^= when ZX
   |)
 where
 boolean X,
 ZX init false;
 end;
```

```
initial equations:
```

$\xi = -1$

```
evolution equations:
```

$\xi' = x + (1 - x^2) * \xi$

```
constraint equations:
```

$x = -zx, zx = \xi * x^2,$

$a^2 = -x - x^2, b^2 = -zx - zx^2$

Intentional Labeled Transition System

A PDS can be viewed as an iLTS $L = (S, Y, I, T)$, where:

- $S \subseteq (\mathbb{Z}/3\mathbb{Z})^n$: set of states,
- Y: set of event variables,
- $I = Sol(Q_0(X))$: set of initial states,
- $T \subseteq S \times \mathbb{Z}/3\mathbb{Z}[Y] \times S$: the symbolic transition relation.

A transition label can be computed directly from PDS by
$P(Y) \equiv Q(s, Y) \oplus (P(s, Y) - s')$

An example

The iLTS of "altern" PDS

$S = \{-1, 0 , 1\}$

$Y = \{a, b, x, zx\}$

$I = \{-1\}$

$T = \{(-1, P_1(Y), 0), \ldots\}$, *where*

$P_1(Y) = (x - (1-x^2)) \oplus (x + zx) \oplus (zx + x^2) \oplus (a^2 + x + x^2) \oplus (b^2 + zx + zx^2)$

Action-based execution

- An infinite (finite) sequence $\sigma = s_0, y_0, s_1, y_1 \ldots$ is an execution if:
  - $s_0 \in I$
  - $\exists P(Y).\ ((s_i, P(Y), s_{i+1}) \in \mathcal{T} \wedge y_i \in Sol(P(Y))), \forall i \in \mathbb{N}$

- The sequence $\sigma_{act} = y_0 y_1 \ldots$ is an action-based execution
- $\|L\|$, $\|L\|_{act}$ denote the sets of executions and action-based executions of an iLTS L, respectively
- Then $\|L\|_{act}$ represents the abstract clock relation semantics of the corresponding synchronous program

Correct transformation

Given two iLTSs L_1, L_2, they have the same semantics if:

$$\forall \sigma_{act}.((\sigma_{act} \in ||L_2||_{act} \Rightarrow \sigma_{act} \in ||L_1||_{act}) \wedge (\sigma_{act} \in ||L_1||_{act} \Rightarrow \sigma_{act} \in ||L_2||_{act}))$$

Refinement

In practice, the requirement is too strong (e.g. program is non-deterministic,…), it should be relaxed as follows:

$$\forall \sigma_{act}.(\sigma_{act} \in ||L_1||_{act} \Rightarrow \sigma_{act} \in ||L_2||_{act})$$

We say that L_1 is a correct transformation of L_2 or L_1 refines L_2, denoted as $L_1 \sqsubseteq L_2$

Symbolic simulation

A symbolic simulation for (L_1, L_2) is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that:

1. for any s_1 if $s_1 \in I_1, \exists s_2 \in I_2, (s_1, s_2) \in \mathcal{R}$
2. for any $(s_1, s_2) \in \mathcal{R}$ it holds:

   if $(s_1, P, s_1') \in T_1$ then there exists a finite set of transitions $(s_2, P_i, s_2^i)_{i \in I} \in T_2$ with $(P \Rightarrow \prod_{i \in I} P_i) \equiv 0 \wedge (q_1', q_2^i) \in \mathcal{R}, \forall i \in I$

Simulation order

$$\begin{array}{ccc} s_1 & \longrightarrow & s_1' \\ \mathcal{R} & & \\ s_2 & & \end{array} \quad \text{can be completed to} \quad \begin{array}{ccc} s_1 & \longrightarrow & s_1' \\ \mathcal{R} & & \color{red}{\mathcal{R}} \\ s_2 & \color{red}{\longrightarrow} & \color{red}{s_2'} \end{array}$$

but not necessarily:

$$\begin{array}{ccc} s_1 & & \\ \mathcal{R} & & \\ s_2 & \longrightarrow & s_2' \end{array} \quad \text{can be completed to} \quad \begin{array}{ccc} s_1 & \color{red}{\longrightarrow} & \color{red}{s_1'} \\ \mathcal{R} & & \color{red}{\mathcal{R}} \\ s_2 & \longrightarrow & s_2' \end{array}$$

Proving refinement by simulation

L_1 is simulated by L_2, denoted as $L_1 \leqslant L_2$, if there exists a symbolic simulation for them.

Let L_1, L_2 be two iLTSs. If there exists a symbolic simulation for (L_1, L_2), then $L_1 \sqsubseteq L_2$ (Soundness)

Symbolic simulation is a preorder, i.e, reflexive and transitive

SIGNAL compiler transformations

```
(| N := FB default (ZN-1)
 | ZN := N$1 init 1
 | FB ^= when (ZN<=1)
 |)
```

```
(| (| CLK_N := ^N
    | CLK_N ^= N
    | ACT_CLK_N{}
    |) |)

  (| CLK_N ^= ZN
   | (| CLK_FB := when (ZN<=1)
      | CLK_FB ^= FB
      | CLK := when (not (ZN<=1))
```

```
(| (| Tick := true
    | when Tick ^= N
    | ACT_Tick{}
    |) |)

   (| when Tick ^= ZN ^= C_FB ^= C
    | (| when C_FB ^= FB |)
       | C_FB := ZN<=1
       | C := not (ZN<=1)
       |)
    |)
```

- → (green) Translations: clock calculation, rewriting to kernel operators,...
- → (red) Optimizations: eliminating sub-expression, trivial clock constraints, clock assignment to generate code, ...

Implementation

- SIGALI is model checker which manipulates polynomial over the finite filed modulo 3
- SIGALI bases on BDD representation to represent polynomial efficiently
- Implement an iterative algorithm to compute the symbolic simulation as an extended library of SIGALI
- Then each transformation of the SIGNAL compiler is followed by our verification process

Conclusion

- A translation validation based verification process.
- Polynomial dynamical systems to represent synchronous programs.
- Formal definition of correct transformation.
- An automated proof method using simulation.
- Application to prove the correctness the SIGNAL compiler transformations.

Thank You!