Low-Level Program Verification using Matching Logic Reachability

Dwight Guth, Andrei Ştefănescu, and Grigore Roşu

University of Illinois at Urbana-Champaign

June 29, 2013

Motivation

- Operational semantics as models of programming languages
- Use operational semantics as basis for
  - interpreters
  - type-checking
  - model-checking
  - deductive program verification

Outline

1. Language Definition
2. Matching Logic Reachability
3. Program Verification

Simple Low Level Language

- implemented in the $\mathbb{K}$ framework
- standard arithmetic and logic operations
- registers
- load/store instructions for memory access
- branching instructions
- interrupts
- I/O instructions
- time units/operation

Basic Instructions

SYNTAX $\quad BInst ::= BOpCode \;\; Register, Exp, Exp$ [strict(3, 4)]

SYNTAX $\quad BOpCode ::= \texttt{add} \mid \texttt{sub} \mid \texttt{mul} \mid \texttt{div} \mid \texttt{or} \mid \texttt{and}$

SYNTAX $\quad Exp ::= Register \mid \#Int$

SYNTAX $\quad Register ::= \texttt{r}Int$

Load/Store Instructions

SYNTAX $\mathit{MInst} ::= \texttt{load}\ \ \mathit{Register}, \mathit{Exp}$ [strict(3)]

SYNTAX $\mathit{MInst} ::= \texttt{store}\ \ \mathit{Exp}, \mathit{Exp}$ [strict(2, 3)]

Branching and Interrupt Instructions

SYNTAX $\mathit{JInst} ::= \texttt{jmp}\ \ \mathit{Id}$

SYNTAX $\mathit{BrInst} ::= \mathit{BrOpCode}\ \ \mathit{Id}, \mathit{Exp}, \mathit{Exp}$ [strict(3, 4)]

SYNTAX $\mathit{BrOpCode} ::= \texttt{beq} \mid \texttt{bne} \mid \texttt{blt} \mid \texttt{ble}$

SYNTAX $\mathit{BrOpCode} ::= \texttt{int}$

SYNTAX $\mathit{NOpCode} ::= \texttt{rfi}$

Sample Program with two Interrupts

```
main: li r0 , #100
      li r1 , #0
      li r2 , #0
      int t1, #7, #10
      int t2, #10, #15
      jmp loop
loop: sub r0 , r0 , #1
      bne loop , r0 , #0
      halt
t1:   add r1 , r1 , #1
      rfi
t2:   add r2 , r2 , #1
      rfi
```

Configuration

CONFIGURATION:

$$\left\langle \begin{array}{l} \langle \mathtt{load}(\$PGM) \curvearrowright \mathtt{jumpTo}(\mathtt{main})\rangle_{\mathsf{k}} \ \left\langle {}^{\cdot}Map \right\rangle_{\mathsf{pgm}} \ \left\langle {}^{\cdot}Map \right\rangle_{\mathsf{mem}} \ \left\langle {}^{\cdot}Map \right\rangle_{\mathsf{reg}} \\ \langle \$TIMING \rangle_{\mathsf{timing}} \ \langle 0 \rangle_{\mathsf{wcet}} \ \langle \$INPUT \rangle_{\mathsf{input}} \ \langle \$INITIAL \rangle_{\mathsf{status}} \ \langle {}^{\cdot}List \rangle_{\mathsf{timers}} \\ \langle 0 \rangle_{\mathsf{priority}} \ \langle {}^{\cdot}List \rangle_{\mathsf{stack}} \ \langle {}^{\cdot}Set \rangle_{\mathsf{interrupts}} \end{array} \right\rangle_{\mathsf{T}}$$

Evaluating Arithmetic Operations

RULE $\left\langle \underset{I2}{\underline{\mathtt{r}I}} \ \cdots \right\rangle_{\mathsf{k}} \ \langle \cdots \ I \mapsto I2 \ \cdots \rangle_{\mathsf{reg}}$

RULE $\left\langle \underset{\mathtt{time}(\mathtt{add})}{\underline{\mathtt{add}\ \ \mathtt{r}I, I2, I3}} \ \cdots \right\rangle_{\mathsf{k}} \ \left\langle \underset{R[I2 + I3 / I]}{\underline{R}} \right\rangle_{\mathsf{reg}}$

Evaluating `load`/`store`

$$\textbf{RULE} \quad \left\langle \frac{\texttt{load}\ \ \texttt{r}l, l2}{\texttt{time(load)}} \cdots \right\rangle_{\mathsf{k}} \ \langle \cdots\ l2 \mapsto l3\ \cdots \rangle_{\mathsf{mem}} \ \left\langle \frac{R}{R[l3/l]} \right\rangle_{\mathsf{reg}}$$

$$\textbf{RULE} \quad \left\langle \frac{\texttt{store}\ \ l, l2}{\texttt{time(store)}} \cdots \right\rangle_{\mathsf{k}} \ \left\langle \frac{M}{M[l2/l]} \right\rangle_{\mathsf{mem}}$$

Evaluating Branching Instructions

RULE $\left\langle \dfrac{\texttt{bne}\ \ X, I, I2}{\texttt{time}(\texttt{bne}) \curvearrowright \texttt{branch}(I \neq I2, X)} \cdots \right\rangle_{\mathsf{k}}$

RULE $\dfrac{\texttt{branch}(\mathsf{true}, X)}{\texttt{jumpTo}(X)}$

RULE $\dfrac{\texttt{branch}(\mathsf{false}, \_)}{\cdot_K}$

Evaluating int

$$\text{RULE} \quad \left\langle \underbrace{\texttt{int}\ \ X, I, I2}_{\texttt{time}(\texttt{int})} \cdots \right\rangle_{\mathsf{k}} \left\langle \cdots \underbrace{\cdot List}_{(X, I + Time, I2)} \right\rangle_{\mathsf{timers}} \langle Time \rangle_{\mathsf{wcet}}$$

`int` schedules an interrupt to fire *I* cycles after executing, and then every *I2* cycles thereafter. The `timers` cell stores the currently activated interrupts in a list of tuples.

Evaluating rfi

RULE $\left\langle \dfrac{\texttt{rfi} \curvearrowright \_}{\texttt{time(rfi)} \curvearrowright K} \right\rangle_{\text{k}} \left\langle \dfrac{(K, Priority)}{\cdot_{List}} \cdots \right\rangle_{\text{stack}} \left\langle \dfrac{\_}{Priority} \right\rangle_{\text{priority}}$

Restore the previously executing code from the `stack` cell, which also contains the previously-executing priority to restore to the `priority` cell. Interrupts are assigned numeric priority in the order they are scheduled by the program, and can interrupt only code running at a lower priority. The main program begins executing at priority 0.

I/O Instructions

- `read`/`write` from a number of buses
- each time cycle, the value on each bus is updated by an external environment

Time Elapsing

RULE $\left\langle \frac{\texttt{time}(O)}{\texttt{waitFor}(\mathit{Timing}(O))} \cdots \right\rangle_{\mathsf{k}} \; \langle \mathit{Timing} \rangle_{\mathsf{timing}}$

RULE
$$\left(\left\langle \frac{\texttt{waitFor}(I)}{\texttt{updateStatus}(I2) \curvearrowright \texttt{updateTimers}(L) \curvearrowright \texttt{interrupt}(L, \texttt{lengthList}\,L)} \cdots \right\rangle_{\mathsf{k}} \; \left\langle \frac{I2}{I2 + I} \right\rangle_{\mathsf{wcet}} \; \left\langle \frac{L}{\cdot_{\mathit{List}}} \right\rangle_{\mathsf{timers}} \right)$$

Each instruction takes a particular number of cycles. Afterwards, the I/O buses are updated and interrupts may fire.

Outline

Matching Logic

- Logic for specifying *static properties* of program configuration and reasoning about them (generalizes separation logic)
- Extends first-order logic with *patterns*
  - Special predicates which are configuration terms with variables
  - Configurations satisfy patterns iff they match them
- Parametric in a model of program configurations (which is axiomatized)

Matching Logic Reachability Rules (ICALP'12, OOPSLA'12, LICS'12)

- "Rewrite" rules over matching logic patterns:

$$\varphi \Rightarrow \varphi'$$

- Semantics: any concrete configuration satisfying φ and terminating reaches a configuration satisfying φ', in the transition system induced by the operational semantics
- Since patterns generalize terms, matching logic reachability rules capture term rewriting rules

Operational Semantics and Axiomatic Semantics as Reachability Rules

- Operational semantics rule $l \Rightarrow r \texttt{ if } b$ is syntactic sugar for reachability rule $l \wedge b \Rightarrow r$
- Hoare triple encoded in a reachability rule with the empty code in the right-hand-side

Reachability Logic

- Language-independent proof system for deriving sequents of the form

$$\mathcal{A} \vdash_{C} \varphi \Rightarrow \varphi'$$

- $\mathcal{A}$(axioms) and C(circularities) are sets of eachability rules
- Intuitively, *symbolic execution* with operational semantics + reasoning with *cyclic behaviors*

Proof System for Reachability

Axiom :

$$\frac{\varphi \Rightarrow \varphi' \quad \in \mathcal{A}}{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi'}$$

Transitivity :

$$\frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2 \qquad \mathcal{A} \cup C \vdash \varphi_2 \Rightarrow \varphi_3}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_3}$$

Reflexivity :

$$\frac{\cdot}{\mathcal{A} \vdash_\emptyset \varphi \Rightarrow \varphi}$$

Circularity :

$$\frac{\mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi'}$$

Logic Framing :

$$\frac{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi' \qquad \psi \text{ is a (patternless) FOL formula}}{\mathcal{A} \vdash_C \varphi \wedge \psi \Rightarrow \varphi' \wedge \psi}$$

Consequence :

$$\frac{\models \varphi_1 \rightarrow \varphi_1' \qquad \mathcal{A} \vdash_C \varphi_1' \Rightarrow \varphi_2' \qquad \models \varphi_2' \rightarrow \varphi_2}{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi_2}$$

Case Analysis :

$$\frac{\mathcal{A} \vdash_C \varphi_1 \Rightarrow \varphi \qquad \mathcal{A} \vdash_C \varphi_2 \Rightarrow \varphi}{\mathcal{A} \vdash_C \varphi_1 \vee \varphi_2 \Rightarrow \varphi}$$

Abstraction :

$$\frac{\mathcal{A} \vdash_C \varphi \Rightarrow \varphi' \qquad X \cap FreeVars(\varphi') = \emptyset}{\mathcal{A} \vdash_C \exists X\, \varphi \Rightarrow \varphi'}$$

Traditional vs Our Approach

- Traditional proof systems: *language-specific*

$$\frac{\{\psi \wedge \mathsf{e} \neq 0\}\, \mathsf{s}\, \{\psi\}}{\{\psi\}\, \mathtt{while}(\mathsf{e})\, \mathsf{s}\, \{\psi \wedge \mathsf{e} = 0\}}$$

- Our proof system: language-independent

Circularity :

$$\frac{\mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{\mathcal{A} \vdash_{C} \varphi \Rightarrow \varphi'}$$

Transitivity :

$$\frac{\mathcal{A} \vdash_{C} \varphi_1 \Rightarrow \varphi_2 \qquad \mathcal{A} \cup C \vdash \varphi_2 \Rightarrow \varphi_3}{\mathcal{A} \vdash_{C} \varphi_1 \Rightarrow \varphi_3}$$

Soundness and Completeness

- Sound (partial correct) with respect to the transition system induced by the semantics
- Relatively complete under some weak assumtions about the configuration model (it can express Gödel β function)
- Proofs size comparable with Hoare logic (FM'12)

Outline

Verifier for a Low-level Language

- Derives program specifications from the operational semantics using the proof system
- Implemented in the $\mathbb{K}$ framework as a set of rules added to the operational semantics
- Reasoning required by the Consequence proof rule
  - Maude, for structural matching
  - Z3, for arithmetic constraints
- Automated (the user only provides the specifications)

Sample program properties

- Upper bounds for the total number of cycles simple programs take to execute (computing the sum of the first "n" numbers, sorting an array, etc)
- Correctness of programs manipulating I/O buses
- Upper bound for the number of cycles a program with interrupts takes to terminate

Sample Program with two Interrupts

```
main: li r0 , M:Int
      li r1 , #0
      li r2 , #0
      int t1, #7, #10
      int t2, #10, #15
      jmp loop
loop: sub r0 , r0 , #1
      bne loop , r0 , #0
      halt
t1:   add r1 , r1 , #1
      rfi
t2:   add r2 , r2 , #1
      rfi
```

Invariant

RULE

$$\left(\begin{array}{c} \left\langle \frac{\$}{\cdot K} \right\rangle_{\text{k}} \ \langle \$ \rangle_{\text{pgm}} \ \langle 0 \rangle_{\text{priority}} \ \langle \cdot List \rangle_{\text{stack}} \\ \left\langle 0 \mapsto \frac{N}{0} \ 1 \mapsto \frac{R1}{\left(R1 + \max\left(0, \left\lceil \frac{(D - T1 + Time)}{10} \right\rceil\right)\right)} \ 2 \mapsto \frac{R2}{\left(R2 + \max\left(0, \left\lceil \frac{(D - T2 + Time)}{15} \right\rceil\right)\right)} \right\rangle_{\text{reg}} \\ \langle \cdots \ \texttt{add} \mapsto 1 \ \ \texttt{rfi} \mapsto 2 \ \cdots \rangle_{\text{timing}} \ \left\langle \frac{Time}{Time + D} \right\rangle_{\text{wcet}} \ \left\langle (\text{t1}, \frac{T1}{\_}, 10) \ (\text{t2}, \frac{T2}{\_}, 15) \right\rangle_{\text{timers}} \end{array}\right) \quad \text{when}$$

$$N > 0 \wedge T1 > Time \wedge T2 > Time \wedge D > 0 \wedge D = 3 * N + 1 + \max\left(0, 3 * \left(\left\lceil \frac{(D - T1 + Time)}{10} \right\rceil\right)\right) + \max\left(0, 3 * \left(\left\lceil \frac{(D - T2 + Time)}{15} \right\rceil\right)\right)$$

Invariant

- Invariant derives `pgm` and `k` cell contents from placement in program
- Invariant depends on timing parameters: side condition uses integer 3
- Current time is Time
- N remaining loop iterations
- All remaining loop iterations plus interrupts last D cycles
- Next interrupts occur at T1 and T2
- Invariant depends on timer frequency: 10 and 15 in denominators
- Priority and stack derived from invariant beginning in normal code
- Number of remaining interrupts derived from fixed-point equations

Conclusions

- $\mathbb{K}$ definition of a low-level language
- Matching logic verifier constructed from the $\mathbb{K}$ definition
- Proofs of upper bound of the number of execution cycles and of correctness