Context-Updates Synthesis and Refinement in Chisel

Irina Mariuca Asavoae, Mihail Asavoae, Adrian Riesco

Namur, Belgium
LOPSTR 2017
Mindset of Chisel

**GENERIC PROGRAM SLICING**, i.e., programming language independent

**GENERATED FROM SEMANTICS SPECIFICATIONS**, e.g., Maude’s rewriting-based algebraic programming language specifications

appeared on Facebook wall under the title “the ultimate hipster fashion”

In futuristic work we could use, for example, K’s formal semantics executable programming language specifications
Gentle Intro in Semantics Based Program Slicing
Gentle Intro in Semantics Based Program Slicing

* The scene:
The victim character:

\[ \text{Sem} : \] the semantics of the programming language given as a rewriting-based algebraic specification (in Maude).

"Rewriting Logic Semantics Project"
Jose Meseguer and Grigore Rosu

The programs are terms of a specific sort \( \text{pgm: P} \) that produce changes to the memory term \( \text{mem: M} \).
Algebraic Specifications of Program Semantics

- \( \text{Sem} \) - snapshot of a few rules defining constructs as:
  - While _ Do _, Local _, and Call _
    - (loop)
    - (local variables)
    - (function call)

\[
\begin{align*}
\text{crl} \ [WR1] & : \langle \text{While} \ b \ \text{Do} \ C, \ st, \ rw, \ fs \rangle \Rightarrow \langle \text{skip}, \ st', \ rw, \ fs \rangle \ \text{if} \ \langle b, \ st \rangle \Rightarrow \langle F, \ st' \rangle. \\
\text{crl} \ [WR2] & : \langle \text{While} \ b \ \text{Do} \ C, \ st, \ rw, \ fs \rangle \Rightarrow \langle \text{skip}, \ st'', \ rw', \ fs \rangle \\
& \ \text{if} \ \langle b, \ st \rangle \Rightarrow \langle T, \ st' \rangle \ \land \ < C ; (\text{While} \ b \ \text{Do} \ C), \ st', \ rw, \ fs \rangle \Rightarrow \langle \text{skip}, \ st'', \ rw', \ fs \rangle. \\
\text{crl} \ [Loc] & : \langle \text{Local} \ vl, \ st, \ rw, \ fs \rangle \Rightarrow \langle \text{skip}, \ st', \ rw, \ fs \rangle \ \text{if} \ \text{st}!' := \text{alloc}(vl, \ st). \\
\text{crl} \ [CFn] & : \langle \text{Call} \ fn \ (\text{actPrms}), \ st, \ rw, \ fs \rangle \Rightarrow \langle \text{skip}, \ st', \ rw', \ fs \rangle \\
& \ \text{if} \ \text{fn} \ (\text{Prms}) \ {\ C} \ \text{fs}!' := \text{fs} /\ \text{lenv} := \text{asgp}(\text{actPrms}, \ \text{Prms}, \ st \ | \ st) /\ \\
& \ < C, \ st \ | \ \text{lenv}, \ rw, \ fs \rangle \Rightarrow \langle \text{skip}, \ st' \ | \ \text{lenv}', \ rw', \ fs \rangle,
\end{align*}
\]
Algebraic Specifications of Program Semantics

\( Sem \) - snapshot of a few rules defining constructs in assembly:

\[
\begin{align*}
\text{addiu } & \_ , \_ , \_ ; \quad \text{jr } \_ ; \quad \text{load } \_ , \_ ; \quad \text{jr } \_ \\
\text{(addition)} & \quad \text{(jump to location)} \quad \text{(memory load)} \quad \text{(function call)}
\end{align*}
\]
Gentle Introduction in Semantics Based Slicing

- $\text{Sem}$ - instance given via one $\text{pgm}$ which contains the procedures/functions $\text{Main}(\,)$, $\text{A}(x, y)$, $\text{B}(x, y)$, $\text{Add}(a, b)$, and $\text{Inc}(z)$ in both high-level and assembly languages semantics.
Algebraic Specifications of Program Semantics

- $Sem$ - executions of a program $pgm$: 

  \[
  \begin{array}{c}
  \text{cfg}[pgm] \\
  t_1, \ldots, t_n = \text{structural variants}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{Kripke structure} \\
  \text{Transition system}
  \end{array}
  \]
Algebraic Specifications of Program Semantics

- $Sem$ - executions of a program $pgm$:

Executions of $pgm$ are traces of $Sem$ in the ground, i.e., memory.
The static analysis perspective:

- *Generic Semantics Based Verification Mindset*
Generic Semantics Based Verification Mindset

Matching & Reachability Logic

invariants, assertions
Generic Semantics Based Program Slicing

* The lead character:

- **Sly**: the **slicer of pgm programs** that are executed based on **Sem**.
**The Leader of the Pride**

- **Sly** - takes a term (of $Sem$) representing the program $pgm : \mathcal{P}$ and, based on the synthesis results obtained by $Syn(i)$, slices $pgm$ producing the $\text{sliced-}pgm$, i.e., the skeleton term of $pgm$ (described in iFM’14):

```plaintext
Main () {
    sum := 0;
    Local i, j;
    i := 1; j := -1;
    While i < 11 Do
        Call A (sum, i);
        Call B (sum, j);
        Call A (j, i)
    }
    A (x, y) {
        If x > 1 Then
            Call Add(x, y);
            Call Inc (y)
        }
    B (x, y) {
        If x > 0 Then
            Call B(x + y, y)
        }
    Add (a, b) {
        a := a + b
    }
    Inc (z) {
        Local i, j;
        i := 1; j := i;
        Call Add (z, i);
        Call Inc (j)
    }
}
Based on the method introduced by Susan Horwitz, Thomas W. Reps, David Binkley in
Interprocedural Slicing Using Dependence Graphs. PLDI 1988

Main() {
    sum := 0;
    Local i, j;
    i := 1; j := -1;
    While i < 11 Do
        Call A(sum, i);
        Call B(sum, j);
        Call A(i, i)
    }

    A(x, y) {
        If x > 1 Then
            Call Add(x, y);
            Call Inc(y)
    }

    B(x, y) {
        If x > 0 Then
            Call B(x+y, y)
    }

    Add(a, b) {
        a := a + b
    }

    Inc(z) {
        Local i, j;
        i := 1; j := i;
        Call Add(z, 1);
        Call Inc(j)
    }
* Chisel is generically slicing programs \( pgm \) based on the programming language semantics;

* Semantics based compilation (see ongoing/future work in \( K \)) is slicing the programming language semantics to make the execution of the program \( pgm \) more efficient.
Generic Semantics Based Program Slicing

* The protagonists:

\[ \text{Syn}(i) : \text{the } i\text{-feature synthesiser of the semantics } Sem. \]
Slicing-Features Synthesis of Sem.

Syn(\(\text{Sem}\)) - synthesizes the \(i\)-feature of the semantics specification \(\text{Sem}\)
and produces language constructs (terms of sort \(P\)) that are directly
responsible for producing the \(\bar{i}\)-feature in the programming language:
Slicing-Features Synthesis of Sem.

Syn(side-effects) - extracts the side-effect constructs,
i.e., the program constructs (terms of sort \( \mathcal{P} \)) that are directly responsible for changes in the memory (identified by the sort \( \mathcal{M} \)):
Slicing-Features Synthesis of Sem.

\textit{Syn}(%side-effects%) - extracts the \textit{side-effect} constructs, i.e., the program constructs (terms of sort \( \mathcal{P} \)) that are \textit{directly} responsible for \textit{changes in the memory} (identified by the sort \( \mathcal{M} \)):

\( \_ := \_ , \_ += \_ \)

\( \text{Call } \_ \)

\( \ldots \)

\( \text{l}w \_ \_ , \text{sw} \_ \_ , \text{add} \_ \_ , \ldots \)

Described in WADT’12 & LOPSTR’15.
So... who is $\text{Syn}(\text{context-updates})$?

- $\text{Syn}(\text{context-updates})$ - extracts the context-update constructs, i.e., the program constructs (terms of sort $\mathcal{P}$) that are directly responsible for stacking-up subparts of memory (stack-ops of $\mathcal{M}$):
Slicing-Features Synthesis of Sem.

\textit{Syn(context-updates)} - extracts the context-update constructs, i.e., the program constructs (terms of sort $P$) that are \textit{directly} responsible for \textit{stacking-up subparts of memory} (stack-ops of $M$):

- Call _
- Throw _
- ...
- j _, jr _, jal _, jalr _

To be described in LOPSTR'17.
So... who is Syn(context-updates)?

* Syn(context-updates): the synthesis discovering context-updates
So... who is Syn(context-updates)?

* Syn(context-updates) : the synthesis discovering context-updates
So... who is Syn(context-updates)?

* Syn(context-updates) : the synthesis discovering context-updates
Memento of Sem

crl [WhileR1] :
  < While be Do C, st, rwb, fs > => < skip, st, rwb, fs >
if < be, st > => < F, st >.

crl [WhileR2] :
  < While be Do C, st, rwb, fs > => < skip, st', rwb', fs >
if < be, st > => < T, st > /
  < C ; (While be Do C), st, rwb, fs > => < skip, st', rwb', fs >.

crl [CallF] :
  < Call fn(actPrms), st, rwb, fs > => < skip, st'', rwb', fs >
if fn(Prms){ C } fs' := fs /
  st' := asgP(actPrms, Prms, st | mt) /
  < C, st', rwb, fs > => < skip, st'' | lenv', rwb', fs >.

crl [LocV] :
  < Local vl, st, rwb, fs > => < skip, st', rwb, fs >
if st' := allocateLocals(vl, st) .
**Memento of Sem’s pgm executions**

- **Sem** - symbolic executions of a program *pgm*:

- **cfg[pgm]**
- **t₁, t₂, t₃, t₄, ..., tₙ** = structural variants

© Grigore Rosu’s presentations
Memento of Sem’s pgm executions

Sem - symbolic executions of a program pgm:
Unification HyperTree of $\text{Syn}(\text{context-updates})$
Unification HyperTree of Syn(context-updates)

The rules (identified by their unique labels) that unify with the configuration terms

\[ \langle \text{pgm}:P, \text{mem}:M \rangle : C, \]

i.e., the rules of form

\[(c)\text{rl [label]} \langle \text{pgm}:P, \text{mem}:M \rangle = \Rightarrow \langle \text{pgm}':P, \text{mem}':M \rangle (i.f) \ldots \]
crl [WhileR1] :
   < While be Do C, st, rwb, fs > => < skip, st, rwb, fs >
   if < be, st > => < F, st > .

crl [WhileR2] :
   < While be Do C, st, rwb, fs > => < skip, st', rwb', fs >
   if < be, st > => < T, st > /
   < C ; (While be Do C), st, rwb, fs > => < skip, st', rwb', fs > .

Unification HyperTree of Syn(context-updates)
Unification HyperTree of Syn\(\text{context-updates}\)
Unification HyperTree of Syn(context-updates)

\[ \text{op asgP : Exp\,Var\,Est} \to \text{Est} . \]
\[ \text{eq } \text{[asgP1]} : \ldots \]
\[ \text{ceq } \text{[asgP2]} : \ldots \]
\[ \text{eq } \text{[asgP3]} : \ldots \]
\[ \text{op } _\text{`(`)}`_\{\} : \text{FunId\,Var\,Com} \to \text{Fun} [\text{ctor}] . \]
\[ \text{op } \text{nf} : \to \text{FunSet} [\text{ctor}] . \]
\[ \text{op } \_\_ : \text{FunSet\,FunSet} \to \text{FunSet} [\text{ctor\comm\assoc\id: nf}] . \]
\[ \text{op } _\text{|_| : Est\,Est} \to \text{Est} [\text{ctor\assoc}] . \]

\[ \text{crl } \text{[CallF]} : \]
\[ < \text{Call fn(actPrms), st, rwb, fs} > \to < \text{skip, st', rwb', fs} > \]
\[ \text{if } \text{fn(Prms)}\{\ C } \text{fs'} := \text{fs} \setminus \text{st'} := \text{asgP(actPrms, Prms, st | mt)} \setminus \]
\[ < C, \text{st', rwb, fs} > \to < \text{skip, st' | lenv', rwb', fs} > . \]
Red label propagation from the memory stack operator _|_ to the CallF rule and further to the Call _ _ language construct (the _P_ subterm of the lefthand side of the rule CallF)
Syn(context-updates) classifies all the red labeled language constructs, e.g., Call _, as elements of \( \mathcal{O} \) - the over-approximated set of context-updates.
Benchmarking Slicing and more...

* PapaBench tasks with their dependencies:
Transforming the program term $\text{pgm}_P$ into a list $L_P$:
Testing the Benchmarks for Synthesis Refinement

Transforming the program term $pgm:P$ into a list $L_P$:

- obtained via $\text{GIT}(pgm)$
- i.e., granular inorder traversal
- with the granularity given by the language constructs sort $C$
Testing the Benchmarks for Synthesis Refinement

* Fragmenting the list $L_p$ into function segments $[L_p]_{fn}$:

```plaintext
Main () {
  sum := 0;
  Local i, j;
  i := 1; j := -1;
  While i < 11 Do
    Call A (sum, i);
    Call B (sum, j);
    Call A (j, i)
  }
A (x, y) {
  If x > 1 Then
    Call Add (x, y);
    Call Inc (y)
  }
B (x, y) {
  If x > 0 Then
    Call B (x + y, y)
  }
Add (a, b) {
  a := a + b
}
Inc (z) {
  Local i, j;
  i := 1; j := i;
  Call Add (z, i);
  Call Inc (j)
}
```
Testing the Benchmarks for Synthesis Refinement

* Using Adrian Riesco’s Maude Testing Tool:

```plaintext
Main () { A (x, y) { Add (a, b) { 
```

**Definition 1.** The property \( \varphi \) w.r.t. \( E \) is defined as follows:

\[
\forall \varsigma \in \mathcal{O}, \forall \omega \in E, \pi := \text{filter}_C(\omega), \forall i \in 1..|\pi| : \pi_i = \varsigma \Rightarrow \\
(\pi_{i-1}\pi_i \in \mathcal{L}_p \Rightarrow \varsigma \in \mathcal{O}_r) \land \\
(\pi_{i-1}\pi_i \notin \mathcal{L}_p \land (\pi_{i-1}, \pi_i) \in [\mathcal{L}_p]_{fn}) \Rightarrow \varsigma \in \mathcal{O}_g) \land \\
(\pi_{i-1}\pi_i \notin \mathcal{L}_p \land (\pi_{i-1}, \pi_i) \notin [\mathcal{L}_p]_{fn}) \Rightarrow \varsigma \in \mathcal{O}_f
\]
Benchmarking Evaluation on Refined Slicing

* PapaBench slicing results:

<table>
<thead>
<tr>
<th>Name</th>
<th># Funs</th>
<th># Calls</th>
<th>LOC</th>
<th>red (%)</th>
<th>LOC</th>
<th>red (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(WhileFun)</td>
<td>(WhileFun)</td>
<td>(MIPS)</td>
<td>(MIPS)</td>
</tr>
<tr>
<td>scheduler_fbw</td>
<td>14</td>
<td>18</td>
<td>103</td>
<td>72.8 %</td>
<td>396</td>
<td>44.4 %</td>
</tr>
<tr>
<td>periodic_auto</td>
<td>21</td>
<td>80</td>
<td>225</td>
<td>73.3 %</td>
<td>779</td>
<td>36.3 %</td>
</tr>
<tr>
<td>fly_by_wire</td>
<td>41</td>
<td>110</td>
<td>638</td>
<td>91.1 %</td>
<td>1913</td>
<td>41 %</td>
</tr>
<tr>
<td>T1</td>
<td>10</td>
<td>26</td>
<td>119</td>
<td>76.5 %</td>
<td>534</td>
<td>36.2 %</td>
</tr>
<tr>
<td>T2</td>
<td>9</td>
<td>9</td>
<td>59</td>
<td>69.5 %</td>
<td>329</td>
<td>44.4 %</td>
</tr>
<tr>
<td>T3</td>
<td>9</td>
<td>24</td>
<td>82</td>
<td>76.5 %</td>
<td>501</td>
<td>43.6 %</td>
</tr>
<tr>
<td>T4</td>
<td>9</td>
<td>14</td>
<td>50</td>
<td>61.5 %</td>
<td>235</td>
<td>34.5 %</td>
</tr>
<tr>
<td>T5</td>
<td>7</td>
<td>22</td>
<td>66</td>
<td>67 %</td>
<td>453</td>
<td>51 %</td>
</tr>
<tr>
<td>autopilot</td>
<td>95</td>
<td>214</td>
<td>1384</td>
<td>92 %</td>
<td>5639</td>
<td>41.5 %</td>
</tr>
<tr>
<td>T6</td>
<td>36</td>
<td>71</td>
<td>306</td>
<td>77.2 %</td>
<td>1329</td>
<td>54 %</td>
</tr>
<tr>
<td>T7</td>
<td>9</td>
<td>13</td>
<td>57</td>
<td>70 %</td>
<td>426</td>
<td>42 %</td>
</tr>
<tr>
<td>T8</td>
<td>7</td>
<td>15</td>
<td>54</td>
<td>69.2 %</td>
<td>219</td>
<td>38 %</td>
</tr>
<tr>
<td>T9</td>
<td>15</td>
<td>30</td>
<td>87</td>
<td>75 %</td>
<td>617</td>
<td>36.5 %</td>
</tr>
<tr>
<td>T10</td>
<td>18</td>
<td>27</td>
<td>102</td>
<td>71.1 %</td>
<td>1002</td>
<td>42.2 %</td>
</tr>
<tr>
<td>T11</td>
<td>3</td>
<td>2</td>
<td>15</td>
<td>63.4 %</td>
<td>90</td>
<td>70.6 %</td>
</tr>
<tr>
<td>T12</td>
<td>4</td>
<td>3</td>
<td>49</td>
<td>66.2 %</td>
<td>363</td>
<td>50 %</td>
</tr>
<tr>
<td>T13</td>
<td>37</td>
<td>93</td>
<td>240</td>
<td>79.7 %</td>
<td>1535</td>
<td>42 %</td>
</tr>
</tbody>
</table>
Contributions, Future and Related Work
Contributions, Future and Related Work

ORBS

Test-based-filtering(context-updates)

There is no spoon

slicer of Maude/K model-checking traces

Syn(context-updates)
Thanks! & Question?