Locally Abstract, Globally Concrete Semantics

A Painless Introduction By Dr. Hähnle

Reiner Hähnle
Software Engineering Group
A Trace Semantics for Concurrent Programs

**Input:** a concurrent program $P$, an initial state $\sigma$

**Output:** all traces $\tau$ of $P$ starting in $\sigma$
Wanted!

A Trace Semantics for Concurrent Programs

Input: a concurrent program $P$, an initial state $\sigma$
Output: all traces $\tau$ of $P$ starting in $\sigma$

Design goals

- Suitable as a semantics for a program verifier
- Modular in the sense of:
  - can add new language concepts incrementally
  - can evaluate locally one statement at a time
  - can constructively and incrementally generate finite/initial traces
# Existing Solutions

<table>
<thead>
<tr>
<th>Method</th>
<th>Author(s)</th>
<th>Complexity</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOS</td>
<td>Plotkin et al.</td>
<td>not modular</td>
<td></td>
</tr>
<tr>
<td>Transition traces</td>
<td>Brookes</td>
<td>very complex, infinite objects</td>
<td></td>
</tr>
<tr>
<td>Action traces</td>
<td>Brookes</td>
<td>complex, limited coverage</td>
<td></td>
</tr>
<tr>
<td>Hybrid traces w/ explicit heap</td>
<td>Kamburjan</td>
<td>complex, infinite objects</td>
<td></td>
</tr>
</tbody>
</table>
The Challenge

\[ m(x) \{ \text{body} \} // \text{Executing asynchronously called method} \]
The Challenge

m(x) { body } // Executing asynchronously called method

We don’t know the context needed for local evaluation

- Caller?
- Processor (executing object)?
- Parameter values?
- Destiny?
The Challenge

m(x) { body }  // Executing asynchronously called method

We don’t know the context needed for local evaluation

- Caller ?
- Processor (executing object) ?
- Parameter values ?
- Destiny ?

s_0 ; s_i  // Composite statement

How to evaluate the s_i incrementally?
Unknown context represented with symbolic expressions
Unknown context represented with symbolic expressions

Parametric Execution Traces

**Symbolic state:** Partial mapping from variables to symbolic expressions
Unknown context represented with symbolic expressions

Parametric Execution Traces

Symbolic state: Partial mapping from variables to symbolic expressions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$X$</td>
</tr>
<tr>
<td>$X$</td>
<td>$*$</td>
</tr>
<tr>
<td>$Y$</td>
<td>$X$</td>
</tr>
<tr>
<td>$z$</td>
<td>$x + 1 + 1$</td>
</tr>
</tbody>
</table>

value expression is symbolic variable (caps)
symbolic variables have unknown value
no chains
maximally evaluated
The Approach (I)

Unknown context represented with symbolic expressions

Parametric Execution Traces

Symbolic state: Partial mapping from variables to symbolic expressions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>*</td>
</tr>
<tr>
<td>Y</td>
<td>*</td>
</tr>
<tr>
<td>z</td>
<td>X + 2</td>
</tr>
</tbody>
</table>

value expression is symbolic variable (caps)  
symbolic variables have unknown value  
no chains  
maximally evaluated
The Approach (I)

Unknown context represented with symbolic expressions

Parametric Execution Traces

Symbolic state: Partial mapping from variables to symbolic expressions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$X$</td>
</tr>
<tr>
<td>$X$</td>
<td>$*$</td>
</tr>
<tr>
<td>$Y$</td>
<td></td>
</tr>
<tr>
<td>$z$</td>
<td>$X + 2$</td>
</tr>
</tbody>
</table>

value expression is symbolic variable (caps)
symbolic variables have unknown value
no chains
maximally evaluated

Traces over symbolic states with path conditions
Unknown context represented with symbolic expressions

Parametric Execution Traces

Symbolic state: Partial mapping from variables to symbolic expressions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>*</td>
</tr>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>X + 2</td>
</tr>
</tbody>
</table>

value expression is symbolic variable (caps)
symbolic variables have unknown value
no chains
maximally evaluated

Traces over symbolic states with path conditions

\[ pc_0 \triangleright \tau_0 = \{ Y_0 > 0 \} \triangleright \langle [x_0 \mapsto Y_0 + 42, \ Y_0 \mapsto \ast] \rangle \bowtie [x_0 \mapsto 17, \ Y_0 \mapsto \ast] \]
How to ensure that incrementally produced traces are well-formed?

For example: a method must have been called before its body is executed.
How to ensure that incrementally produced traces are well-formed?

For example: a method must have been called before its body is executed.

Events

Order-sensitive concurrent statements inject events into traces

\[ \tau \triangleright \text{invEv(mtdName, value)} \]
The Approach (II)

How to ensure that incrementally produced traces are well-formed?

For example: a method must have been called before its body is executed

Events

Order-sensitive concurrent statements inject events into traces

$$\tau \rightsquigarrow invEv(mtdName, value)$$

Well-Formedness

$$wf(\tau \rightsquigarrow invREv(m, v)) = wf(\tau) \land \#_\tau(invEv(m, v)) > \#_\tau(invREv(m, v))$$
How to evaluate composite statements incrementally?

Continuations

Traces end with a **continuation marker** containing a statement:

\[ pc \triangleright \tau \cdot \lambda(s) \in CTr \]

**Meaning:** traces of \( s \) must still be generated and appended to \( \tau \)
1. For each statement in the language design a local evaluation rule:

\[ \text{val}_\sigma : \text{Stmt} \rightarrow 2^{\text{Ctr}} \]
1. For each statement in the language design a local evaluation rule:

$$\text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}}$$

2. Design a trace composition rule extending traces and continuations:
   Given concrete trace $sh$ and continuation $\lambda(s)$:
1. For each statement in the language design a **local** evaluation rule:

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

2. Design a **trace composition rule** extending traces and continuations:

Given concrete trace \( sh \) and continuation \( \lambda(s) \):
- Evaluate first statement of \( s \) to \( \tau \) in state \( \text{last}(sh) \), with \( \lambda(s') \) remaining
How to Design an LAGC Semantics in 4 Easy Steps?

1. For each statement in the language design a local evaluation rule:

   \[ \text{val}_\sigma : \text{Stmt} \rightarrow 2^{\text{Ctr}} \]

2. Design a trace composition rule extending traces and continuations:
   Given concrete trace \( sh \) and continuation \( \lambda(s) \):
   - Evaluate first statement of \( s \) to \( \tau \) in state \( \text{last}(sh) \), with \( \lambda(s') \) remaining
   - Stitch \( sh \) and \( \tau \) together, ensure well-formedness, continuation now \( \lambda(s') \)
1. For each statement in the language design a **local** evaluation rule:

   \[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

2. Design a **trace composition rule** extending traces and continuations:
   Given **concrete** trace \( sh \) and continuation \( \lambda(s) \):
   - Evaluate first statement of \( s \) to \( \tau \) in state \( \text{last}(sh) \), with \( \lambda(s') \) remaining
   - Stitch \( sh \) and \( \tau \) together, ensure **well-formedness**, continuation now \( \lambda(s') \)

3. Design **well-formedness** predicate ensuring proper event order
1. For each statement in the language design a local evaluation rule:

   \[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

2. Design a trace composition rule extending traces and continuations:
   Given concrete trace \( sh \) and continuation \( \lambda(s) \):
   - Evaluate first statement of \( s \) to \( \tau \) in state \( \text{last}(sh) \), with \( \lambda(s') \) remaining
   - Stitch \( sh \) and \( \tau \) together, ensure well-formedness, continuation now \( \lambda(s') \)

3. Design well-formedness predicate ensuring proper event order

4. To obtain global trace semantics of program \( P \):
   Exhaustively apply (2.) starting with \( \langle \sigma_{\text{Init}} \rangle \) and continuation \( \lambda(P) \)
The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

\[ \text{val}_\sigma(\text{skip}) = \{ \emptyset \upharpoonright \langle \sigma \rangle \cdot \lambda(\emptyset) \} \]
The local evaluation rules

$$\text{val}_\sigma : \text{Stmt} \rightarrow 2^{\text{Ctr}}$$

$$\text{val}_\sigma (x := e) = \{ \emptyset \triangleright \langle \sigma \rangle \bowtie \sigma [x \mapsto \text{val}_\sigma (e)] \cdot \lambda (\emptyset) \}$$
The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

\[ \text{val}_\sigma (\text{if } e \{ s \}) = \{ \{ \text{val}_\sigma (e) = \text{tt} \} \triangleright \langle \sigma \rangle \cdot \lambda (s), \} \]
The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

\[ \text{val}_\sigma(\text{if } e \{ s \}) = \{ \{ \text{val}_\sigma(e) = \text{tt} \} \triangleright \langle \sigma \rangle \cdot \lambda(s), \{ \text{val}_\sigma(e) = \text{ff} \} \triangleright \langle \sigma \rangle \cdot \lambda(\emptyset) \} \]
The local evaluation rules

\[
\text{val}_\sigma : \text{Stmt} \rightarrow 2^{\text{CTr}}
\]

\[
\text{val}_\sigma(r; s) = \{ pc \triangleright \tau \cdot \lambda(r') \in \text{val}_\sigma(r) \}
\]
The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{CTr}} \]

\[ \text{val}_\sigma (r; s) = \{ pc \triangleright \tau \cdot \lambda(r'; s) \mid pc \triangleright \tau \cdot \lambda(r') \in \text{val}_\sigma (r) \} \]
The local evaluation rules

\[ \text{val}_\sigma : \text{Stmt} \to 2^{\text{Ctr}} \]

The trace composition rule (\(\sigma\) concrete)

\[ \sigma = \text{last}(sh) \]

\[ sh, \lambda(s) \]
The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

The trace composition rule (\(\sigma\) concrete)

\[
\begin{align*}
\sigma &= \text{last}(sh) \\
\text{pc} &\triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \\
sh, \lambda(s) &\in \text{val}_\sigma(s)
\end{align*}
\]
The local evaluation rules

\[ \text{val}_\sigma : \text{Stmt} \rightarrow 2^{\text{Ctr}} \]

The trace composition rule (\(\sigma\) concrete)

\[
\sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \quad pc \text{ consistent}
\]

\[ sh, \lambda(s) \]
Case Study 1: A While Language

The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

The trace composition rule (\(\sigma\) concrete)

\[
\begin{align*}
\sigma &= \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \quad pc \text{ consistent} \\
sh, \lambda(s) &\rightarrow sh \ast\ast \tau, \lambda(s')
\end{align*}
\]
Case Study 1: A While Language

The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

The trace composition rule

\[ \sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \quad pc \text{ consistent} \]

\[ sh, \lambda(s) \rightarrow sh \ast\ast \tau, \lambda(s') \]

The well-formedness predicate: Nada
The local evaluation rules

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

The trace composition rule

\[
\begin{align*}
\sigma &= \text{last}(sh) \\
\text{pc} \triangleright \tau \cdot \lambda(s') &\in \text{val}_\sigma(s) \\
\text{pc consistent} \\
sh, \lambda(s) &\rightarrow sh ** \tau, \lambda(s')
\end{align*}
\]

The well-formedness predicate: Nada

The global trace semantics for program \( P \)

\[ \langle [x \rightarrow 0], \ x \in \text{vars}(P) \rangle, \ \lambda(P) \]
Case Study 1: A While Language

The **local evaluation rules**

\[ \text{val}_\sigma : Stmt \rightarrow 2^{\text{Ctr}} \]

The **trace composition rule**

\[
\begin{align*}
\sigma &= \text{last}(sh) \\
\text{pc} &\uparrow \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \\
\text{pc} &\text{ consistent}
\end{align*}
\]

\[
sh, \lambda(s) \rightarrow sh \star \star \tau, \lambda(s')
\]

The **well-formedness predicate**: Nada

The **global trace semantics** for program \( P \)

\[
\langle [x \rightarrow 0], x \in \text{vars}(P) \rangle, \lambda(P) \rightarrow sh_1, \lambda(s_1)
\]
The local evaluation rules

\[ \text{val}_\sigma : Stmt \to 2^{\text{Ctr}} \]

The trace composition rule

\[
\sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \quad pc \text{ consistent} \\
sh, \lambda(s) \to sh \star \tau, \lambda(s')
\]

The well-formedness predicate: Nada

The global trace semantics for program \( P \)

\[
\langle [x \to 0], x \in \text{vars}(P) \rangle, \lambda(P) \to sh_1, \lambda(s_1) \star \to sh_n, \lambda(\emptyset)
\]
Case Study 1: A While Language

The local evaluation rules

\[ \operatorname{val}_\sigma : \text{Stmt} \rightarrow 2^{\text{Ctr}} \]

The trace composition rule

\[
\sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \operatorname{val}_\sigma(s) \quad pc \text{ consistent} \\
sh, \lambda(s) \rightarrow sh \star \star \tau, \lambda(s')
\]

The well-formedness predicate: Nada

The global trace semantics for program \( P \)

\[
\langle [x \rightarrow 0], x \in \text{vars}(P) \rangle, \lambda(P) \rightarrow sh_1, \lambda(s_1) \xrightarrow{*} \lim_{i \rightarrow \infty} sh_i
\]
Discussion

Observations

- No need to represent intermediate states in rules of composite statements
- Exactly one trace generated—only one path condition can be consistent (Expected: while-language is deterministic)
- Programs allowed to diverge
- Good match with program logic based on symbolic execution
Discussion

- No symbolic traces needed
- No events needed
- No well-formedness needed
Case Study 2: Procedure Calls

The local evaluation rules

Call a parallel procedure \( m \) with parameter \( e \):

\[
\text{val}_\sigma(\text{call}(m, e)) = \{\emptyset \triangleright \text{invEv}_\sigma(m, \text{val}_\sigma(e)) \cdot \lambda(\emptyset)\}
\]
The local evaluation rules

Begin to execute a parallel procedure $m$:

$$\text{val}_\sigma(m(x)\{s\}) = \{ \emptyset \triangleright inv\text{REv}_\sigma(m, Y) \bowtie \sigma[y \mapsto Y, Y \mapsto \star] \cdot \lambda(s[x \leftarrow y]) \mid Y \not\in \text{dom}(\sigma) \}$$
The local evaluation rules

Begin to execute a parallel procedure $m$:

$$\text{val}_\sigma(m(x)\{s\}) = \begin{cases} \emptyset \triangleright invREv_\sigma(m, Y) \bowtie \sigma \left[ Y \mapsto * \right] \cdot \lambda(s[x \leftarrow y]) \\ y, Y \notin \text{dom}(\sigma) \end{cases}$$
The local evaluation rules

Begin to execute a parallel procedure $m$:

$$\text{val}_\sigma(m(x)\{s\}) = \{ \emptyset \triangleright \text{invREv}_\sigma(m, Y) \bowtie \sigma[y \mapsto Y, \ Y \mapsto \ast] \cdot \lambda(s[x \leftarrow y]) | \ y, \ Y \notin \text{dom}(\sigma) \}$$
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

Continuations of parallel procedures selected from task pool (multiset) \( q \)

\[
\sigma = \text{last}(sh)
\]

\[
sh, \ q + \{\lambda(s)\}
\]
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

Continuations of parallel procedures selected from task pool (multiset) \( q \)

\[
\sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \quad pc \text{ consistent}
\]

\[
sh, \ q + \{\lambda(s)\}
\]
The local evaluation rules

The trace composition rules

Continuations of parallel procedures selected from task pool (multiset) $q$

\[
\sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(s) \quad pc \text{ consistent}
\]

\[
sh, \ q + \{\lambda(s)\} \rightarrow sh \ast \ast \tau, \ q + \{\lambda(s')\}
\]
The local evaluation rules

The trace composition rules

Starting new task:

\[ m(x)\{s\} \in \bar{M} \]

\[ \sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(m(x)\{s\}) \]

\[ sh, q \rightarrow \]
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

Starting new task: concretise method evaluation $\tau$

\[ m(x)\{s\} \in \overline{M} \quad \rho \text{ concretises } \tau \quad \rho(pc) \text{ consistent} \]
\[ \sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(m(x)\{s\}) \]
\[ sh, q \rightarrow \rho(sh) \star\star \rho(\tau) \]
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

Starting new task: concretise method evaluation $\tau$ and extend task pool $q$

$$m(x)\{s\} \in \overline{M} \quad \rho \text{ concretises } \tau \quad \rho(pc) \text{ consistent}$$

$$\sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(m(x)\{s\})$$

$$\quad sh, q \rightarrow \rho(sh) \ast \ast \rho(\tau), q + \{\lambda(s')\}$$
The local evaluation rules

The trace composition rules

Starting new task: concretise method evaluation $\tau$ and extend task pool $q$

\[
m(x)\{s\} \in \overline{M} \quad \rho \text{ concretises } \tau \quad \rho(pc) \text{ consistent} \\
\sigma = \text{last}(sh) \quad pc \triangleright \tau \cdot \lambda(s') \in \text{val}_\sigma(m(x)\{s\}) \quad \text{wf}(\rho(sh)**\rho(\tau)) \\
sh, q \rightarrow \rho(sh)**\rho(\tau), q+\{\lambda(s')\}
\]
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

The well-formedness predicate

\[ wf(sh \rightsquigarrow invREv(m, v)) = wf(sh) \land \#_{sh}(invEv(m, v)) > \#_{sh}(invREv(m, v)) \]
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

The well-formedness predicate

The global trace semantics for program $P$

$$\langle I_P \rangle, \lambda(P) \xrightarrow{*} sh, n \cdot \lambda(\emptyset)$$
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

The well-formedness predicate

The global trace semantics for program $P$

$$\langle I_P \rangle, \lambda(P) \xrightarrow{*} \lim_{i \to \infty} sh_i$$
Case Study 2: Procedure Calls

The local evaluation rules

The trace composition rules

The well-formedness predicate

The global trace semantics for program $P$

Observations

- Local evaluation rules could be reused
- Minor modification to trace composition rule
- One new trace composition rule
Case Study 3: ABS
Won’t fit on one slide — read the paper!
The most modular concurrent semantics the world has ever seen?

- Presented version has statement-level process interleaving
  \[\Rightarrow\text{Atomic-by-default easily possible (as in ABS)}\]
- Rules for nearly full ProMeLa, ABS in paper
- Dynamic logic calculus with soundness proof for while-language
- Executable Isabelle mechanization of most features by N. Heidler
What is LAGC?

Lewd Artists Getting Canceled
Literally Arrogant General Computer Scientists
Louisiana Associated General Contractors (www.lagc.org)
Left Autoservo Gyro Control
Lazily Aggregated Gradient Coding
Lower Austrian Geo Cachers
Lovemaking Androids Grope Compulsively
Linguistically Advanced Grammatically Constrained
Laterally Askance Geometric Corpus
Ludo And his Geeky Colleagues
Liberally Applied Gherkin Cream
Last Australian Golf Club