Verification of Parallel Programs with the Owicki-Gries and Rely-Guarantee Methods in Isabelle/HOL

Leonor Prensa Nieto
TU München

Marseille, 7 January 2002
Overview

• Motivation.
• Hoare logic for parallel programs.
  – The Owicki-Gries method.
  – The rely-guarantee method.
• Formalization in Isabelle/HOL.
• Completeness for parameterized parallel programs.
• Application, examples.
• Conclusion.
Motivation

- Parallel programs appear in safety critical applications.
- Verification is necessary, sometimes difficult and mostly tedious.
- Techniques:
  1. Testing.
  2. Model Checking.
  3. Interactive theorem provers: PVS, Coq, Isabelle/HOL ...
Hoare Logic for Parallel Programs

• 1965, Dijkstra introduces the \texttt{parbegin} statement.
• 1969, Hoare proposes a formal system of axioms and inference rules for the verification of imperative sequential programs.
• 1976, Susan Owicki and David Gries extend Hoare’s system for the verification of parallel programs with shared variables.
• 1981, Cliff Jones introduces the rely-guarantee method, a compositional version of the Owicki-Gries system.
Hoare Logic

- Hoare triples have the form $\{P\} \; c \; \{Q\}$
- A Hoare triple is valid, i.e. $\models \{P\} \; c \; \{Q\}$ iff every execution starting in a state satisfying $P$ ends up in a state satisfying $Q$ (partial correctness).
- Hoare logic $\equiv$ inference rules for deriving valid Hoare triples.

\[\vdash \{Q[e/x]\} \; x := e \; \{Q\} \quad (Assign)\]

\[\begin{align*}
\vdash \{P\} \; c_0 \; \{M\} \\
\vdash \{M\} \; c_1 \; \{Q\}
\end{align*} \quad (Sequence)\]
Soundness and Completeness

- The system is **sound** if all the specifications that are derivable are also valid

\[ \vdash \{P\} c \{Q\} \implies \models \{P\} c \{Q\} \]

- The system is **complete** if all specifications that are valid can be derived

\[ \models \{P\} c \{Q\} \implies \vdash \{P\} c \{Q\} \]
Compositionality

- Hoare logic is compositional for sequential programs.
- Disjoint parallel programs

\[
\begin{align*}
\{ \text{True} \} & \quad \{ y=0 \} \quad \{ y=0 \} \\
\text{x:=0} & \quad \text{y:=3} \quad \implies \quad \text{x:=0 || y:=3} \\
\{ x=0 \} & \quad \{ y=3 \} \quad \{ x=0 \wedge y=3 \}
\end{align*}
\]

\implies \text{Compositional}

- But if \( c_1 \) and \( c_2 \) share variables, then there is no operator \( \text{Op} \) such that in general

\[
\begin{align*}
\{ P_1 \} & \quad \{ P_2 \} \quad \{ \text{Op} (P_1, P_2) \} \\
\text{c}_1 & \quad \text{c}_2 \quad \nRightarrow \quad \text{c}_1 || \text{c}_2 \\
\{ Q_1 \} & \quad \{ Q_2 \} \quad \{ \text{Op} (Q_1, Q_2) \}
\end{align*}
\]

\implies \text{Not compositional}
Example

These two programs have the same behaviour when executed sequentially:

\[ x := x + 2 \iff x := x + 1; x := x + 1 \]

but they deliver different results when composed in parallel with for example the program "\( x := 0 \)":

\[
\begin{align*}
\{ x = 0 \} & \quad \{ x = 0 \} \\
\text{x:=0 || x:=x+2} & \quad \text{x:=0 || x:=x+1; x:=x+1} \\
\{ x = 0 \lor x = 2 \} & \quad \{ x = 0 \lor x = 1 \lor x = 2 \}
\end{align*}
\]

\[ \Rightarrow \text{Not compositional (because of interference)} \]
The Owicki-Gries Method

- First complete logic for proving correctness of parallel programs with shared variables.
- Component programs are specified as proof outlines which are free from interference.

<table>
<thead>
<tr>
<th>Pre-post specification</th>
<th>Proof outlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x=0} \xrightarrow{\text{x:=x+1;}} x:=x−1 \xrightarrow{\text{x:=x+1;}} {x=0}</td>
<td>{x=0} \xrightarrow{x:=x+1;} x:=x−1 \xrightarrow{\text{x:=x+1;}} {x=1} \xrightarrow{x:=x−1} {x=0}</td>
</tr>
</tbody>
</table>

Proof outlines have the property that whenever the execution of a program reaches an assertion with state $\sigma$, this assertion is true of that state.
Interference Freedom

Given two proof outlines

\[ P_1: \{ p_1 \} \]
\[ c_1 \]
\[ \{ p_2 \} \]
\[ c_2 \]
\[ \vdots \]
\[ P_2: \{ q_1 \} \]
\[ a_1 \]
\[ \{ q_2 \} \]
\[ a_2 \]
\[ \vdots \]

We say that they are interference free iff

\[ \forall p_i \in \text{assertions of } P_1 \land \forall a_j \in \text{atomic actions of } P_2, \]
\[ \{ p_i \land \text{pre } a_j \} \]
\[ a_j \]
\[ \{ p_i \} \]

(and vice versa)

Note: If \( P_1 \) has \( n \) statements and \( P_2 \) has \( m \) statements, proving interference freedom requires proving \( O(n \times m) \) correctness formulas.
Example

These two proof outlines are correct but not interference free. For example, the assertion $x=0$ is not preserved against the atomic action $x:=x+2$:

$\{x=0\} \parallel \{x=0\}$

$\{x=0\}$

By weakening the postconditions we obtain both correct and interference free proof outlines:

$\{x=0\} \parallel \{x=0\} \implies x:=x+2 \parallel x:=0$

$\{x=0 \lor x=2\} \parallel \{x=0 \lor x=2\}$

$\{x=0 \lor x=2\}$
Rule for Parallel Composition

\[
\begin{array}{c}
\{P_1\} c_1 \{Q_1\}, \ldots, \{P_n\} c_n \{Q_n\} \text{ are correct and interference-free} \\
\{P_1 \land \ldots \land P_n\} c_1 \parallel \ldots \parallel c_n \{Q_1 \land \ldots \land Q_n\}
\end{array}
\]

This rule is not compositional, i.e. a change in one of the components may affect the proof, not only of the modified component, but also of all the others.
The Rely-Guarantee Method

\[ \models P \text{ sat } (\text{pre, rely, guar, post}) \]

\( P \) satisfies its specification if under the \textbf{assumptions} that

1. \( P \) is started in a state that satisfies \textit{pre}, and
2. any environment transition in the computation satisfies \textit{rely},

then \( P \) ensures the following \textbf{commitments}:

3. any component transition satisfies \textit{guar}, and
4. if the computation terminates, the final state satisfies \textit{post}. 
Rule for Parallel Composition

\[
\begin{align*}
(rely \lor guar_1) &\rightarrow rely_2 \\
(rely \lor guar_2) &\rightarrow rely_1 \\
(guar_1 \lor guar_2) &\rightarrow guar \\
c_1 \text{ sat } (pre, rely_1, guar_1, post_1) \\
c_2 \text{ sat } (pre, rely_2, guar_2, post_2) \\
\hline 
\end{align*}
\]

\[
c_1 \parallel c_2 \text{ sat } (pre, rely, guar, post_1 \land post_2)
\]

- Advantages over Owicki-Gries:
  1. Compositional.
  2. Lower complexity.
Formalization in Isabelle

- The Programming Language
  - Abstract Syntax
  - Operational Semantics
- Proof Theory
  - Proof System
  - Soundness
The Programming Language

\[ c_1 \parallel \cdots \parallel c_n \]

The component programs \( c_i \) are sequential while-programs with synchronization.

- **Syntax:** (\( \alpha \) represents the state and is an argument of the program)
  \[ \alpha \text{ com} = \text{Basic} (\alpha \Rightarrow \alpha) \]
  \[ | (\alpha \text{ com}); (\alpha \text{ com}) \]
  \[ | \text{IF } (\alpha \text{ bexp}) \text{ THEN } (\alpha \text{ com}) \text{ ELSE } (\alpha \text{ com}) \text{ FI} \]
  \[ | \text{WHILE } (\alpha \text{ bexp}) \text{ INV } (\alpha \text{ assn}) \text{ DO } (\alpha \text{ com}) \text{ OD} \]
  \[ | \text{AWAIT } (\alpha \text{ bexp}) \text{ THEN } (\alpha \text{ com}) \text{ END} \]

\[ \alpha \text{ par\_com} = (\alpha \text{ com}) \text{ list} \]

- **Interleaving semantics**
  \[
  \frac{(Ts[i], s) \rightarrow^1 (r, t)}{(Ts, s) \rightarrow^1 (Ts[i := r], t)}
  \]
Parameterized Parallel Programs

Many interesting parallel programs are given schematically in terms of a parameter $n$, representing the number of components. For example,

\[
\begin{align*}
\{ x = 0 \} \\
\big\| \big|_{i=0}^{n} x &:= x + 1 \\
\{ x = n + 1 \}
\end{align*}
\]

Syntax: $\|^{n}_{i=0} c~i \equiv c~0 \| c~1 \| \ldots \| c~n$

In HOL the “\ldots” can be expressed by the function map and a list from 0 to $n$, i.e.

\[
\text{map} \ (\lambda i. \ c \ i) \ [0..n]
\]
Completeness for parameterized parallel programs

Generalized rule for the Owicki-Gries system:

\[ \forall i \leq n. \vdash \{ P(i, n) \} \ c(i, n) \ \{ Q(i, n) \} \]
\[ \forall i, j \leq n. i \neq j \rightarrow \text{the proofs outlines of} \]
\[ \{ P(i, n) \} c(i, n) \{ Q(i, n) \} \text{ and } \{ P(j, n) \} c(j, n) \{ Q(j, n) \} \]
\[ \text{are interference free} \]

\[ \vdash \{ \bigcap_{i=0}^{n} P(i, n) \} \parallel \bigcap_{i=0}^{n} c(i, n) \ \{ \bigcap_{i=0}^{n} Q(i, n) \} \]

We have mechanically proven with Isabelle that this system is sound, but is it complete?

We know (by the completeness of the non-parameterized systems) that for each value of \( n \), we can find a derivation in the system but ... can we find for every valid specification of a parameterized program a single derivation that works for all values of \( n \)?

We have proven that the answer is yes.
Application

- Nice external syntax
- Automatic generation of the verification conditions
- Examples
Syntax

\{x=0 \land y=0\} \iff \{(x,y). x=0 \land y=0\}

y:=y+2 \iff \text{Basic } \lambda(x, y). (x, y+2)

c_1 \parallel \cdots \parallel c_n \iff [c_1, \cdots, c_n]

- Representation of program variables via the quote/antiquote technique.
The inferences rules and axioms are systematically applied backwards until all verification conditions are generated.
## Examples

### Owicki-Gries:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Verif. cond.</th>
<th>Automatic</th>
<th>Lemmas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peterson</td>
<td>122</td>
<td>122</td>
<td>0</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>20</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>Ticket (param)</td>
<td>35</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>Zero search</td>
<td>98</td>
<td>98</td>
<td>0</td>
</tr>
<tr>
<td>Prod/Cons</td>
<td>138</td>
<td>125</td>
<td>3</td>
</tr>
</tbody>
</table>

### Rely-Guarantee:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Spec.lines</th>
<th>Verif. Cond</th>
<th>Lemmas</th>
<th>Proof Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single mutator garbage collector</td>
<td>35</td>
<td>289</td>
<td>28</td>
<td>408</td>
</tr>
<tr>
<td>Multi-mutator garbage collector (param)</td>
<td>35</td>
<td>328</td>
<td>36</td>
<td>756</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Verif. cond.</th>
<th>Lemmas</th>
<th>Proof Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set array to 0 (param)</td>
<td>8</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>Increment variable (param)</td>
<td>14</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>Find least element in array (param)</td>
<td>22</td>
<td>1</td>
<td>30</td>
</tr>
</tbody>
</table>
Conclusion

• First formalization of the Owicki-Gries and rely-guarantee methods in a general purpose theorem prover.
• Improvements over the original formalizations:
  – No need for program locations.
  – Support for schematic programs.
• Special interest in applicability: concrete syntax, automation, examples ...
• New completeness proof for parameterized parallel programs.
• Tool useful not only for the “a posteriori” verification, but also in the search for a proof.