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 **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.

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

$$\frac{\vdash \{P\}\ c_0\ \{M\} \quad \vdash \{M\}\ c_1\ \{Q\}}{\vdash \{P\}\ c_0;\ c_1\ \{Q\}}\ (Sequence)$$

$$\vdots$$

Soundness and Completeness

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

$$\vdash \{P\}\ c\ \{Q\} \Longrightarrow\ \models \{P\}\ c\ \{Q\}$$

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

$$\models \{P\}\ c\ \{Q\} \Longrightarrow\ \vdash \{P\}\ c\ \{Q\}$$

Compositionality

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

$$\begin{array}{llcl} \{\text{True}\} & \{y=0\} & & \{y=0\} \\ x:=0 & y:=3 & \Longrightarrow & x:=0 \parallel y:=3 \\ \{x=0\} & \{y=3\} & & \{x=0 \wedge y=3\} \end{array}$$

$\Longrightarrow$ Compositional

- But if c_1 and c_2 share variables, then there is no operator Op such that in general

$$\begin{array}{llcl} \{P_1\} & \{P_2\} & & \{\text{Op}\ (P_1, P_2)\} \\ c_1 & c_2 & \not\Longrightarrow & c_1 \parallel c_2 \\ \{Q_1\} & \{Q_2\} & & \{\text{Op}\ (Q_1, Q_2)\} \end{array}$$

$\Longrightarrow$ Not compositional

Example

These two programs have the same behaviour when executed sequentially:

$$x:=x+2 \Longleftrightarrow x:=x+1;\ x:=x+1$$

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

$\{x=0\}$	$\{x=0\}$
$x:=0 \parallel x:=x+2$	$x:=0 \parallel x:=x+1;\ x:=x+1$
$\{x=0 \vee x=2\}$	$\{x=0 \vee x=1 \vee x=2\}$

$\Longrightarrow$ 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.

Pre-post specification	Proof outlines
$\{x=0\}$	$\{x=0\}$
$x:=x+1;$	$x:=x+1;$
$x:=x-1$	$\{x=1\}$
$\{x=0\}$	$x:=x-1$
	$\{x=0\}$

Proof outlines have the property that whenever the execution of a program reaches an assertion with state σ, this assertion is true of that state.

Interference Freedom

Given two proof outlines

$$\begin{array}{lll} \mathsf{P}_1{:}\{\mathsf{p}_1\} & \qquad & \mathsf{P}_2{:}\{\mathsf{q}_1\} \\ \quad \mathsf{c}_1 & & \quad \mathsf{a}_1 \\ \quad \{\mathsf{p}_2\} & & \quad \{\mathsf{q}_2\} \\ \quad \mathsf{c}_2 & & \quad \mathsf{a}_2 \\ \quad \vdots & & \quad \vdots \end{array}$$

We say that they are interference free iff

$$\boxed{\begin{array}{c} \forall \mathsf{p}_i \in \text{assertions of } \mathsf{P}_1 \wedge \forall \mathsf{a}_j \in \text{atomic actions of } \mathsf{P}_2, \\[1em] \{\mathsf{p}_i \wedge \text{pre } \mathsf{a}_j\} \\ \mathsf{a}_j \\ \{\mathsf{p}_i\} \\[1em] \text{(and vice versa)} \end{array}}$$

Note: If P_1 has n statements and P_2 has m statements, proving interference freedom requires proving $\mathcal{O}(n \times m)$ 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:

$$\begin{array}{lcl} \{x=0\} & & \{True\} \\ x:=x+2 & \| & x:=0 \\ \{x=2\} & & \{x=0\} \end{array} \qquad \begin{array}{c} \{x=0 \wedge x=0\} \\ x:=x+2 \\ \{x=0\} \end{array}$$

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

$$\begin{array}{ccc} \{x=0\} & & \{True\} \\ x:=x+2 & \| & x:=0 \\ \{x=0 \vee x=2\} & & \{x=0 \vee x=2\} \end{array} \qquad \begin{array}{c} \{x=0 \vee x=2 \wedge x=0\} \\ x:=x+2 \\ \{x=0 \vee x=2\} \end{array} \quad \Longrightarrow \quad \begin{array}{c} \{x=0\} \\ x:=x+2 \ \|\ x:=0 \\ \{x=0 \vee x=2\} \end{array}$$

Rule for Parallel Composition

$$\frac{\{P_1\}\ c_1\ \{Q_1\}, \ldots, \{P_n\}\ c_n\ \{Q_n\} \text{ are correct and interference-free}}{\{P_1 \land \ldots \land P_n\}\ c_1 \| \ldots \| c_n\ \{Q_1 \land \ldots \land Q_n\}}$$

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

$$\boxed{\models P \textsf{ sat } (pre,\ rely,\ guar,\ post)}$$

P satisfies its specification if under the assumptions that

1. P is started in a state that satisfies pre, and
2. any environment transition in the computation satisfies $rely$,

then P ensures the following commitments:

3. any component transition satisfies $guar$, and
4. if the computation terminates, the final state satisfies $post$.

Rule for Parallel Composition

$$\frac{\begin{array}{l}(rely \vee guar_1) \longrightarrow rely_2 \\ (rely \vee guar_2) \longrightarrow rely_1 \\ (guar_1 \vee guar_2) \longrightarrow guar \\ c_1\ \textsf{\textit{sat}}\ (pre,\ rely_1,\ guar_1,\ post_1) \\ c_2\ \textsf{\textit{sat}}\ (pre,\ rely_2,\ guar_2,\ post_2)\end{array}}{c_1 \parallel c_2\ \textsf{\textit{sat}}\ (pre,\ rely,\ guar,\ post_1 \wedge post_2)}$$

- 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

$$\mathsf{c}_1 \parallel \cdots \parallel \mathsf{c}_n$$

The component programs c_i are sequential while-programs with synchronization.

- **Syntax:** (α represents the state and is an argument of the program)

  α com = Basic ($\alpha \Rightarrow \alpha$)
  | (α com); (α com)
  | IF (α bexp) THEN (α com) ELSE (α com) FI
  | WHILE (α bexp) INV (α assn) DO (α com) OD
  | AWAIT (α bexp) THEN (α com) END

  α par_com = (α com) list

- **Interleaving semantics**

$$\frac{(Ts[i], s) \longrightarrow^1 (r, t)}{(Ts, s) \longrightarrow^1 (Ts[i := r], t)}$$

Parameterized Parallel Programs

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

$$\{x = 0\}$$
$$\|_{i=0}^{n}\ x := x + 1$$
$$\{x = n + 1\}$$

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

In HOL the "..." can be expressed by the function map and a list from 0 to n, i.e.

$$map\ (\lambda i.\ c\ i)\ [0..n]$$

Completeness for parameterized parallel programs

Generalized rule for the Owicki-Gries system:

$$\frac{\begin{array}{l}\forall i \leq n.\ \vdash \{P(i,n)\}\ c(i,n)\ \{Q(i,n)\} \\ \forall i,j \leq n.\ i \neq j \longrightarrow \textit{the proofs outlines of} \\ \{P(i,n)\}c(i,n)\{Q(i,n)\}\ \textit{and}\ \{P(j,n)\}c(j,n)\{Q(j,n)\} \\ \textit{are interference free}\end{array}}{\vdash \{\bigcap_{i=0}^{n}\ P(i,n)\}\ \|_{i=0}^{n}\ c(i,n)\ \{\bigcap_{i=0}^{n}\ Q(i,n)\}}$$

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

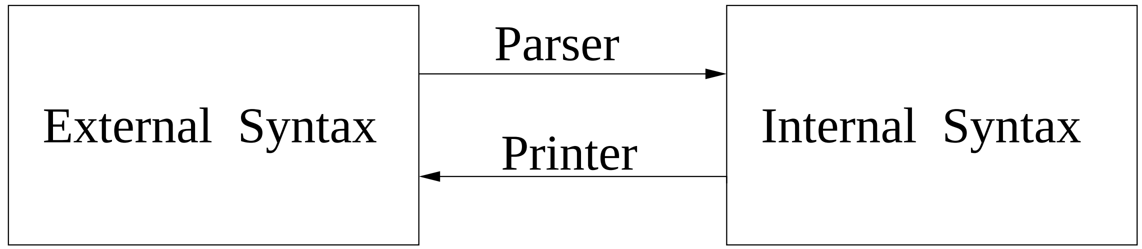
We know (by the completeness of the non-parameterized systems) that for each value of n, we can find a derivation in the system but ... can we find for every valid specification of a parameterized program a single derivation that works for all values of n?

We have proven that the answer is yes.

Application

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

Syntax



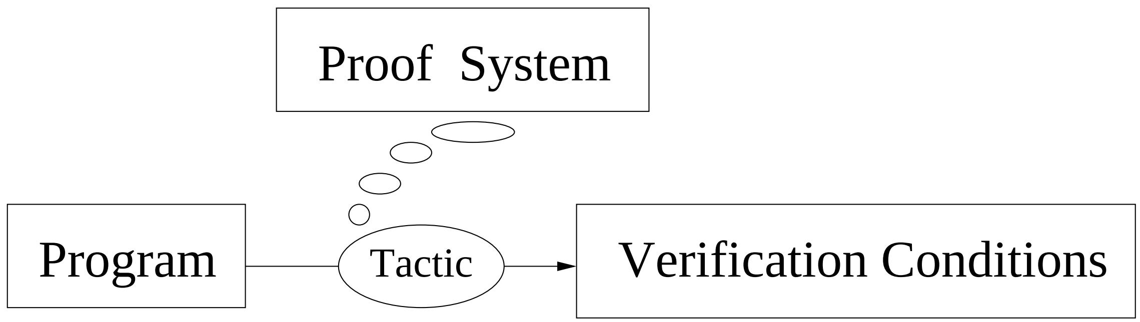
$\{x{=}0 \wedge y{=}0\} \longleftrightarrow \{(x,y).\ x{=}0 \wedge y{=}0\}$

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

$c_1 \parallel \cdots \parallel c_n \longleftrightarrow [c_1, \cdots, c_n]$

- Representation of program variables via the quote/antiquote technique.

Verification Conditions Generation



The inferences rules and axioms are systematically applied backwards until all verification conditions are generated.

Examples

Owicki-Gries:

Algorithm	Verif. cond.	Automatic	Lemmas
Peterson	122	122	0
Dijkstra	20	20	0
Ticket (param)	35	24	0
Zero search	98	98	0
Prod/Cons	138	125	3

Algorithm	Spec.lines	Verif. Cond	Lemmas	Proof Steps
Single mutator garbage collector	35	289	28	408
Multi-mutator garbage collector (param)	35	328	36	756

Rely-Guarantee:

Algorithm	Verif. cond.	Lemmas	Proof Steps
Set array to 0 (param)	8	3	40
Increment variable (param)	14	3	23
Find least element in array (param)	22	1	30

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.