Proving termination of parallel programs

Citation for published version (APA):

Apt, K. R., Boer, de, F. S., & Olderog, E-R. (1990). *Proving termination of parallel programs*. (Computing science notes; Vol. 9013). Technische Universiteit Eindhoven.

Document status and date:

Published: 01/01/1990

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Proving Termination of Parallel Programs

by

K.R.Apt F.S. de Boer E.R.Olderog

90/13

October , 1990

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
Editors: prof.dr.M.Rem
 prof.dr.K.M. van Hee

Proving Termination of Parallel Programs

Krzysztof R. Apt
Centre for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam
The Netherlands
and
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188
U.S.A.

Frank S. de Boer
Technical University Eindhoven
Department of Mathematics and Computer Science
P.O.Box 513
5600 MB Eindhoven
The Netherlands

Ernst-Rüdiger Olderog
Department of Computer Science
University of Oldenburg
2900 Oldenburg
Federal Republic of Germany

In Owicki and Gries [1976] a well known proof method for verifying parallel programs based on the interference freedom test was introduced. We show that their extension of the method to deal with termination is incorrect and suggest two ways of repairing it.

Note: This paper appeared in: "Beauty is Our Business, A Birthday Salute to Edsger W. Dijkstra" (eds. W. Feijen, N. van Gasteren, D. Gries, J. Misra), Springer-Verlag, New York, pages 0-6, 1990.

Keywords and Phrases: program verification, interference freedom, parallelism, termination.

1985 Mathematics Subject Classification: 68N05, 68Q55, 68Q60.

1 Introduction

The Owicki-Gries method [1976] for verifying partial correctness of parallel programs calls for finding interference free proof outlines for partial correctness of component programs. A proof outline for a partial correctness proof of $\{p\} S \{q\}$, where p and q are assertions and S is a program, is a construct of the form $\{p\} S^* \{q\}$, where S^* is the program S annotated with the assertions used in the proof of $\{p\} S \{q\}$. For example, consider the two component programs

```
S1 ≡ while  $x > 0$  do
       $y := 0$ ;
      if  $y = 0$  then  $x := x - 1$  else  $y := 0$  fi
    od
```

and

```
S2 ≡ while  $x > 0$  do
       $y := 1$ ;
      if  $y = 1$  then  $x := x - 1$  else  $y := 1$  fi
    od.
```

Here is a proof outline for $\{\text{true}\} S_1 \{\text{true}\}$ (the one for $\{\text{true}\} S_2 \{\text{true}\}$ is similar):

```
{true}
while  $x > 0$  do
  {true}
   $y := 0$ ;
  {true}
  if  $y = 0$  then {true}
                     $x := x - 1$ 
                    {true}
                else {true}
                     $y := 0$ 
                    {true}
  fi
od
{true}.
```

Proof outlines for component programs are *interference free* if the component programs do not invalidate the assertions in each others' proof outlines. In this case, the proof outlines remain valid annotations when the component programs are executed in parallel. In the above example, showing that S_2 does not invalidate the assertions of the above proof outline for S_1 requires proving

the following. Let r be any assertion in the proof outline for S_1 , let R be any assignment in S_2 , and let $pre(R)$ be the precondition for R in the proof outline for S_2 . Then the following must be proved:

$$\{r \wedge pre(R)\} R \{r\}.$$

Showing interference freedom for the proof outlines for S_1 and S_2 above is trivial, since all assertions in the proof outlines equal **true**. Hence, the proof outlines for S_1 and S_2 are interference free.

To extend the method to total correctness, Owicki and Gries proposed two steps. First, in the usual fashion, associate a bound function with each loop of each component program. A bound function is an integer expression that decreases with each loop iteration and remains non-negative. Clearly, the existence of a bound function ensures that the loop terminates when considered in isolation.

Second, to ensure termination of the parallel execution of the component programs, add the following interference freedom requirement: no component program increases a bound function of a loop of another component program.

Now consider the component programs S_1 and S_2 above. Using x as the bound function for both loops, it is clear that the additional interference freedom requirement is satisfied. And yet, it is also clear that S_1 and S_2 when executed in parallel need not terminate, for they may synchronize in such a fashion that x is never decreased. Hence, the additional interference freedom requirement proposed by Owicki and Gries is not correct.

2 A Solution

The proof of total correctness of a loop requires showing that the bound function is decreased with each iteration. Formally, we can use the following proof rule motivated by Dijkstra [1982] (EWD 573):

WHILE-RULE

$$\frac{\begin{array}{l} \{p \wedge B\} S \{p\}, \\ \{p \wedge B \wedge t = z\} S \{t < z\}, \\ p \wedge B \rightarrow t > 0 \end{array}}{\{p\} \text{ while } B \text{ do } S \text{ od } \{p \wedge \neg B\}}$$

where t is an integer expression and z is an integer variable that does not appear in p, t, B or S .

The first premise states that p is a loop invariant, the second that the bound function t is decreased with each iteration and the third that if another iteration can be performed then t is positive.

If such a loop appears in a component process, then interference freedom should require that the proof of the loop's correctness, using the above rule is not

invalidated. The partial correctness proof outline already includes the necessary assertions concerning the first premise $\{p \wedge B\} S \{p\}$. However, it does not include the assertions concerning the second premise $\{p \wedge B \wedge t = z\} S \{t < z\}$. Returning to our example, it is readily seen that it is this part of the proof of the loop of component S_1 that is falsified by execution of component S_2 . If the assertions from this second assumption are included in the proof outline, then the original interference freedom requirement of Owicki and Gries will suffice.

One way to achieve this is by starting from a modification of this proof rule where the first two premises are replaced with

$$\{p \wedge B \wedge t = z\} S \{p \wedge t < z\}$$

and by introducing the following formation rule for a proof outline for total correctness of **while**-loops.

Definition (Proof Outline I: while-loops)

$$\frac{\begin{array}{l} \{p \wedge B \wedge t = z\} S^* \{p \wedge t < z\}, \\ p \wedge B \rightarrow t > 0 \end{array}}{\{\text{inv} : p\} \text{ while } B \text{ do } \{p \wedge B \wedge t = z\} S^* \{p \wedge t < z\} \text{ od } \{p \wedge \neg B\}}$$

where t is an integer expression, z is an integer variable not occurring in p, t, B or S and $\{p \wedge B \wedge t = z\} S^* \{p \wedge t < z\}$ is a proof outline for total correctness. $\square$

The annotation $\{\text{inv} : p\}$ represents the invariant of the loop **while** B **do** S **od**. Since the bound functions are now absorbed into the assertions, we can drop the condition for interference freedom of the bound functions and simply use the original definition of interference freedom for partial correctness.

With these changes the Owicki-Gries method for verifying total correctness of parallel programs is correct.

A drawback of the above method is that it forces us to mix the proofs of the invariance of p and of the decrease of t . The resulting proof outlines therefore become quite heavy. On the other hand this method provides a close relationship between program annotation and program execution. Since

$$\{p \wedge B\} z := t \{p \wedge B \wedge t = z\},$$

we can expand the conclusion of the above formation rule so that every **while**-loop starts with an assignment $z := t$:

$$\{\text{inv} : p\} \text{ while } B \text{ do } \{p \wedge B\} z := t \{p \wedge B \wedge t = z\} S^* \{p \wedge t < z\} \text{ od } \{p \wedge \neg B\}.$$

With this expansion, the following pleasing property of proof outlines for partial correctness holds again:

Claim Let $\{p_i\} S_i^* \{q_i\}$, $i \in \{1, \dots, n\}$, be interference free expanded proof outlines for total correctness and let S'_i for $i \in \{1, \dots, n\}$ be the program resulting from S_i^* by deleting all assertions but keeping the new assignments of the form $z := t$. Consider an execution of the parallel program $[S'_1 || \dots || S'_n]$ starting in a state satisfying $p_1 \wedge \dots \wedge p_n$. Whenever the control in one of the component programs reaches a point annotated by an assertion, this assertion is true. $\square$

3 Another Solution

Another possibility is to assume that the proof of decrease of t is of a particularly simple form, namely that for a loop body S

- (i) all assignments inside S decrease t or leave it unchanged,
- (ii) on each syntactically possible path through S at least one assignment decreases t .

By a *path* we mean here a possibly empty finite sequence of assignments. Sequential composition $\pi_1; \pi_2$ of paths π_1 and π_2 is lifted to sets Π_1 and Π_2 of paths by putting

$$\Pi_1; \Pi_2 = \{\pi_1; \pi_2 \mid \pi_1 \in \Pi_1 \text{ and } \pi_2 \in \Pi_2\}.$$

By ε we denote the empty sequence. For any path π we have $\pi; \varepsilon = \varepsilon$; $\pi = \pi$.

Definition Let S be a **while**-program. We define the path set of S , denoted by $path(S)$, by induction on the structure of S :

- $path(skip) = \{\varepsilon\}$,
- $path(u := t) = \{u := t\}$,
- $path(S_1; S_2) = path(S_1); path(S_2)$,
- $path(\text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi}) = path(S_1) \cup path(S_2)$,
- $path(\text{while } B \text{ do } S_1 \text{ od}) = \{\varepsilon\}$.

$\square$

Thus, each path through S is identified with the sequence of assignments lying on it. Note that in the last clause we take into account only the case when the loop is terminated immediately. This is sufficient for establishing condition (ii) above.

We define the notion of a *proof outline for total correctness* as for partial correctness, except for the case of **while**-loops for which we use the following formation rule.

Definition (Proof Outline II: while-loops)

- (1) $\{p \wedge B\} S^* \{p\}$,
 - (2) $\{pre(R) \wedge t = z\} R \{t \leq z\}$ for every assignment R within S ,
 - (3) for each path $\pi \in path(S)$ there exists an assignment R in π such that $\{pre(R) \wedge t = z\} R \{t < z\}$,
 - (4) $p \wedge B \rightarrow t > 0$
-
- $\{inv : p\} \{bd : t\} \text{ while } B \text{ do } \{p \wedge B\} S^* \{p\} \text{ od } \{p \wedge \neg B\}$

where t is an integer expression and z is an integer variable not occurring in p, t, B or S^* . Here $\{p \wedge B\} S^* \{p\}$ is a proof outline for total correctness and $pre(R)$ stands for the assertion preceding R in this proof outline. $\square$

The annotation $\{bd : t\}$ represents the bound function of the loop **while** B **do** S **od**. With this new definition of a proof outline for total correctness, the Owicki-Gries method for verifying total correctness of parallel programs is correct.

With this definition we can no longer justify the proof outlines for the component programs used in Section 1. Indeed, along the path $y := 0; y := 0$ of the first loop body the proposed bound function x does not decrease.

Observe that when the empty path ε is an element of $path(S)$, we cannot verify premise (3) of the above rule. Thus it may happen that we can prove total correctness of a **while**-program using the **while**-rule but are unable to record this proof as a proof outline for total correctness. An example is the program

```

b := true;
while b do
  if b then b := false else skip fi
od

```

whose termination can be easily established. This shows some limitations of the above approach to recording proofs of total correctness.

However, various parallel programs can be successfully handled in this way. For example, the bound function given in the proof of termination of program *Findpos* in Owicki and Gries [1976] satisfies the more stringent conditions (2)–(4) given above. This provides a justification for their proof.

Note We discovered, when attempting to prove its soundness, that the original version of the Owicki-Gries method for proving total correctness is incorrect.

□

Acknowledgement Detailed comments of David Gries enabled us to improve the presentation.

References

- [1] E.W. Dijkstra, *Selected Writings on Computing*, Springer-Verlag, New York, 1982.
- [2] S. Owicki and D. Gries, An axiomatic proof technique for parallel programs, *Acta Informatica* 6, 1976, 319–340.

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
87/04	T.Verhoeff	Delay-insensitive codes - An overview.
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns - searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language.
87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J. Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J. Paredaens D. Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J. Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.
88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.

88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.

89/13	A.T.M.Aerts K.M. van Hee M.W.H. Heslen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.
90/1	W.P.de Roever-H.Barringer C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, pp. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems,

p. 17.

90/16 F.S. de Boer
C. Palamidessi

A fully abstract model for concurrent logic languages, p. 23.

90/17 F.S. de Boer
C. Palamidessi

On the asynchronous nature of communication in concurrent logic languages: a fully abstract model based on sequences, p. 29.