



Proceedings of the 16th
**Nordic Workshop on
Programming Theory**

October 6–8, 2004
Uppsala, Sweden

Paul Pettersson and Wang Yi (Eds.)

Preface

The objective of the Nordic Workshop on Programming Theory is to bring together researchers from (but not limited to) the Nordic and Baltic countries interested in programming theory, in order to improve mutual contacts and cooperation.

The 16th Nordic Workshop on Programming Theory took place at the Uppsala University, Sweden, 6-8 October 2004. The previous workshops were held in Uppsala (1989 and 1999), Aalborg (1990), Gothenburg (1991 and 1995), Bergen (1992 and 2000), Turku (1993, 1998, and 2003), Aarhus (1994), Oslo (1996), Tallinn (1997 and 2002), and in Lyngby (2001).

There were 39 regular presentations at the workshop, arranged in two parallel sessions. In addition the following five invited speakers gave presentations in plenary sessions: Erik Hagersten (Uppsala Univ., Sweden), Neil D. Jones (Copenhagen Univ., Denmark), Kim G. Larsen (Aalborg Univ., Denmark), P.S. Thiagarajan (National University of Singapore), and Michael Williams (Ericsson, Sweden).

Programme Committee

Michael R. Hansen	Techn. U. of Denmark, Denmark
Magne Haveraaen	Univ. of Bergen, Norway
Hannu-Matti Jarvinen	Tampere Univ of Tech., Finland
Kim G. Larsen	Aalborg Univ., Denmark
Bengt Nordstrom	Univ. of Gothenburg, Chalmers Univ of Tech., Sweden
Olaf Owe	Univ. of Oslo, Norway
Kaisa Sere	Abo Akademi University, Finland
Tarmo Uustalu	Inst. of Cubernetics, Estonia
Juri Vain	Tallinn Technical University, Estonia
Wang Yi	Uppsala Univ., Sweden

Local Organizing Committee

Ulrika Andersson, Anders Hessel, Patrik Johansson, Paul Pettersson, Wang Yi

Table of Contents

The size-change approach to program termination analysis	
<i>Neil D. Jones</i>	1
Memory Usage Estimation for Java Cards	
<i>Gerardo Schneider</i>	2
Stage-Preserving Embeddings of Languages	
<i>Todd L. Veldhuizen</i>	5
Signals and Comonads	
<i>Tarmo Uustalu and Varmo Vene</i>	9
Simulation-Based Iteration of Tree Transducers	
<i>Parosh Abdulla, Axel Legay, Julien d’Orso, and Ahmed Rezine</i>	11
Heuristic Guided Model-Checking of Real-Time Systems	
<i>Henning Dierks</i>	14
Minimal DBM Substraction	
<i>Johan Håkansson, Kim G. Larsen, and Paul Pettersson</i>	17
Color-blind Behavioral Specifications for Transformations of Reactive Synchronous Programs	
<i>Andrzej Wasowski, Ulrik Larsen, and Kim G. Larsen</i>	21
A formal approach to model-driven engineering	
<i>Dubravka Ilic and Elena Troubitsyna</i>	24
Cofree Coalgebras for Signature Morphisms	
<i>Uwe Wolter</i>	26
Transfinite Corecursion	
<i>Härmel Nestra</i>	29
Scenario(MSC)-Based Specifications: a Survey	
<i>P.S. Thiagarajan</i>	32
Compositional Specification and Verification of UML Models	
<i>Marcel Kyas, and Frank S. de Boer</i>	33
Refining UML interactions	
<i>Ragnhild Kobro Runde</i>	35
An Analysis Tool for UML SPT Models	
<i>John Håkansson and Leonid Mokrushin</i>	38
The Controlled Linear Programming Problem	
<i>Henrik Björklund, Olle Nilsson, Ola Svensson, and Sergei Vorobyov</i> .	44
Programming Languages Capturing Complexity Classes	
<i>Lars Kristiansen och Paul J. Voda</i>	47
Coin Games	
<i>Rafal Somla</i>	51

Automated Black-Box Testing of Functional Correctness using Function Approximation	
<i>Karl Meinke</i>	52
Specifying Test Cases Using Observer Automata	
<i>Johan Blom, Anders Hessel, Bengt Jonsson, and Paul Pettersson</i> . . .	54
Modeling and Simulating an Industrial Robot	
<i>Johan Andersson, Pavel Krčál, Leonid Mokrushin, Christer Norström, Anders Wall, and Wang Yi</i>	57
A simulator approach to data race detection	
<i>Karl Marklund and Björn Victor</i>	60
Encoding the Applied π-calculus in the π-calculus	
<i>Anders Bloch, Morten V. Frederiksen, and Bjørn Haagensen</i>	61
A Timed Mobile Calculus	
<i>Jing Chen</i>	64
Axioms and Algorithms for True Concurrency Equivalences	
<i>Michael Kannellos, and Mila Majster-Cederbaum</i>	67
Slicing CSP-OZ Specifications	
<i>Ingo Brückner</i>	70
Generic implementations of process calculi in Isabelle	
<i>Jesper Bengtsson</i>	73
Inference of Timed Transition Systems	
<i>Olga Grinchtein, Bengt Jonsson, and Martin Leucker</i>	78
Comparative Linear-Time Semantics on Action-Labelled Continuous-Time Markov Chains	
<i>Verena Wolf and Christel Baier</i>	79
A Robust Interpretation of Duration Calculus	
<i>Martin Fränzle and Michael R. Hansen</i>	82
Implementing stronger encapsulation for Java-like languages	
<i>Ville Leppänen and Sami Mäkelä</i>	85
Object Calculus and Self-Application	
<i>Neil Ghani, and Johan Glimming</i>	87
A Hoare Logic for Distributed Objects with Asynchronous Method Calls	
<i>Johan Dovland, Einar Broch Johnsen, and Olaf Owe</i>	89
Optimal Scheduling and Priced Timed Automata	
<i>Kim G. Larsen</i>	92
Abstraction Based Analysis and Arbiter Synthesis: Radar Memory Interface Card Case Study Revisited	
<i>Juhan Ernits</i>	93

Timed Traces and Strand Spaces	
<i>Robin Sharp and Michael R. Hansen</i>	95
Intelligent Sensory Using Compact Data Structures and Bayesian Networks	
<i>Jens A. Hansen</i>	98
Universal Semi-local Election Protocol Using Forward Links	
<i>Dobieslaw Wróblewski</i>	101
Towards programming logics for low-level languages	
<i>Ando Saabas, Gilles Barthe, Lilian Burdy, and Tamara Rezk</i>	104
A Program Inverter LRinv and its Structure	
<i>Masahiko Kawabe and Robert Glück</i>	107
Extensions of Event Based B for Development of Grid Systems	
<i>Pontus Boström and Marina Waldén</i>	109
Sequent Calculi for Temporal Logics of Common Knowledge and Belief	
<i>Jüraté Sakalauskaitė</i>	112

Slicing CSP-OZ Specifications*

Ingo Brückner

Department für Informatik, Universität Oldenburg

26111 Oldenburg, Germany

Fax: +49-441-798-2965

ingo.brueckner@informatik.uni-oldenburg.de

6th September 2004

1 Introduction

The combination of the two well known formal specification techniques CSP [Hoa78, Hoa85] for specification of behavioural aspects of systems and Object-Z (OZ) [Smi00, Spi92, WD96] for specification of data aspects of systems into the specification language CSP-OZ [Fis97] has already been subject of intense research and is currently further extended by an integration with Duration Calculus (DC) [HHF⁺94, HZ97, ZHR91] for specification of real time aspects of systems. This leads to the formalism of CSP-OZ-DC [HO02] which gives rich possibilities to specify any aspect of complex systems with a suitable formalism in an overall coherent way.

An important challenge, especially, when trying to automatically or semi-automatically analyse such system specifications is their inherent complexity which quickly goes beyond the scope of current analysis techniques such as model checking – in spite of the amazing progress that has recently been done in this research area.

In order to tackle this problem on a different level we propose the application of a technique that is already very long and well known in the area of program analysis, namely applying a kind of ‘program slicing’ [Wei81, Tip95, HDZ00] to CSP-OZ-DC specifications. The basic idea is to reduce a given specification by eliminating some of its components in such a way that its semantics remains unchanged w.r.t. a given property under consideration (the slicing criterion).

This reduction is based on a preceding analysis of the specification that is very similar to well known program analysis techniques [ASU97, Muc00, NNH99]. More specifically, it includes the construction of a specification dependence graph which comprises – similar to

a program dependence graph [HRB90] – all relevant kinds of dependences that are present in the specification such as control or data dependences between specification elements.

Our current work is a first step towards the goal of slicing CSP-OZ-DC specifications which is yet restricted to an application to its untimed predecessor CSP-OZ and accordingly untimed properties that are expressed in an untimed DC variant called State/Event Duration Calculus. From this starting point we develop a slicing algorithm that we show to be correct in the initially claimed sense, i.e. from the given property’s point of view the resulting reduced specification exhibits the same behaviour as the original specification does.

2 Specifying with CSP-OZ

We will introduce CSP-OZ by specifying an air condition system as an example. The system consists of several components which communicate with each other. In the example specification we will focus on the actual air condition component. A fragment of its CSP-OZ specification is depicted in figure 1.

The CSP part of the specification defines the air condition’s behaviour.

This is done by introducing a set of channels that define the air condition’s means of communicating with

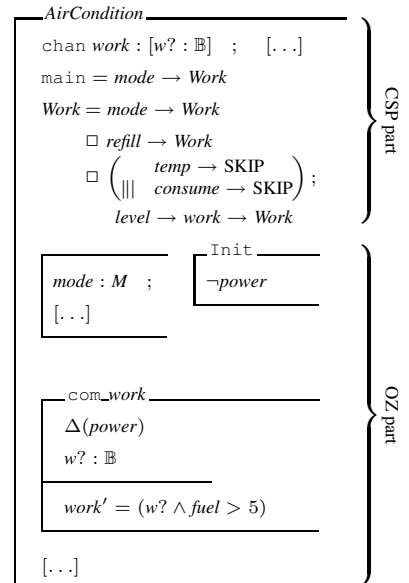


Figure 1: Air condition CSP-OZ specification fragment

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center ‘Automatic Verification and Analysis of Complex Systems’ (SFB/TR 14 AVACS). See www.avacs.org for more information.

its surrounding components. How this communication looks like is defined in several CSP processes that represent different communications the controller can engage in. These may be communications together with or independent of the surrounding components.

The system's state space, its initial configuration and operations on the state space are then defined in the specification's Object-Z part by so called schemas. Each schema consists of two parts: The upper part may contain a list of variables that are defined in the lower part and a list of channels that are used in the lower part for incoming or outgoing communications. The lower part can contain preconditions which determine whether the associated operation is enabled or not and it can contain equations that describe the effect that the associated operation has on the state space. Schemas can be connected to communications in the CSP part but they can also be independent from CSP communications, i.e. the associated operations are then possible at any time.

3 Constructing the Specification Dependence Graph

Following the usual approaches in program slicing [HRB90], a necessary precondition is to analyse the given specification w.r.t. dependences that it contains. The underlying idea is to be afterwards able to backtrack any specification element that might directly or indirectly be relevant for components of the property which serves as the slicing criterion.

Control Flow Graph The first step in constructing the specification dependence graph is to determine the specification's control flow graph which is defined inductively over the structure of the specification's CSP part. Nodes of the control flow graph represent single communications, CSP operators or process calls while edges of the control flow graph represent the possible control flow between the nodes they connect.

Control Dependences Based on the analysis of control dependences between separate specification elements, groups of control flow graph nodes are accumulated into basic blocks, i.e. groups of nodes which are not control dependent on each other. Accordingly, control flow graph edges inside such basic blocks are removed at this point and edges between basic blocks are now regarded as representing control dependences in the sense that the originating node controls the target node.

Data Dependences The last step in constructing the specification dependence graph consists in analysing

the specification w.r.t. data dependences between control flow graph nodes, i.e. dependences due to the definition of a variable in one node and the subsequent reference of the same variable in a different node. Two basic kinds of data dependences are considered: First, direct data dependence which occurs between two nodes that are directly connected by a path in the control flow graph from the first to the second node without an intermediate node with an intervening definition. Second, interference data dependence is considered which occurs between two nodes that are located in different branches of a parallel interleaving CSP operator. Data dependence edges can connect different nodes inside the same basic blocks as well as nodes that are located in different basic blocks.

Specification Dependence Graph The specification dependence graph finally comprises all nodes that were introduced during the control flow graph construction and all edges determined during the control dependence analysis and the data dependence analysis.

4 Slicing

When the previously described preparations have been performed, the actual slicing of the specification is easily done: Initially, a set of marked nodes in the specification dependence graph is determined by analysing which node has direct influence on the property that constitutes the slicing criterion. Direct influence is present if a node's associated schema name appears directly in the property or if a node's associated schema contains a definition of a variable which appears in the property.

After this initialisation, the set of marked nodes is repeatedly augmented with yet unmarked nodes that are the origin of edges which lead to already marked nodes. This backtracking is performed until a fixpoint is reached and the set of marked nodes can not be increased any more by application of this rule. At this point only some further nodes are added to the set of marked nodes. These further nodes have certain types and are located in the same basic block as already marked nodes. They are so called structure nodes which are needed to maintain the wellformedness of the specification slice.

This slice is finally derived from the original specification based on the set of marked nodes in the specification dependence graph: All specification elements that correspond to unmarked nodes can safely be removed from the specification. A last step consists in removing all variables from the specification's state space that are not used inside any schema any more.

In order to formalise this slicing algorithm, as it is informally described here, we first define a formal seman-

tics of CSP-OZ specifications based on a variant of labeled Kripke structures (LKS) [CCO⁺04]. This is done separately for the specification's CSP and for its OZ part which are then combined by parallel composition in order to give the semantics of the full specification.

Further, we need to formalise the properties which serve as slicing criteria. To this end we use State/Event Duration Calculus (SE-DC), a discrete and untimed variant of DC. With SE-DC formulas we can express properties of CSP-OZ specifications as well as properties of the underlying LKS semantics.

Finally, the formalisation of the actual slicing corresponds directly to the informal description given here: Similar to the way the specification slice is derived from the original specification by removing certain specification elements, the specification slice's LKS is related to the original specification's LKS in that certain states and transitions are missing that correspond to the removed specification elements.

5 Slicing Correctness

The presented slicing approach can be regarded as correct if the following holds: The original specification fulfils a property if and only if also the specification slice w.r.t. to this property fulfils this property.

In the correctness proof, which at this point is still work in progress, we plan to apply two steps: First, a definition of stuttering equivalence [Lam83] of LKSs w.r.t. a set of atomic propositions and events is given and it is shown that the presented slicing algorithm guarantees stuttering equivalence between the original specification's LKS and the specification slice's LKS w.r.t. to a set of atomic propositions and events derived from the property that served as a slicing criterion. In a second step it is shown that this kind of stuttering equivalence between two LKSs is equivalent to the notion of correctness that we stated above.

6 Conclusion

This contribution proposes the application of the well known technique of program slicing in a different area, namely in the specification of behavioural and data aspects of complex systems with CSP-OZ. It is shown which adaptations are needed when transferring the existing approaches of program analysis to the new application field of specification analysis.

Furthermore the correctness of the proposed algorithm is shown based on a formally defined semantics of the specification language and an appropriate logic for expressing specification properties.

Future steps will include the extension of the approach to timed specifications (CSP-OZ-DC), and tool

support for automatically performing the slicing of such specifications. The latter will definitely be needed when dealing with larger case studies that are foreseen in the project AVACS [AVA04] which forms the overall context of this work.

References

- [ASU97] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley, 1997.
- [AVA04] AVACS. Transregional collaborative research center 14 avacs: Automatic verification and analysis of complex systems. 2004.
- [CCO⁺04] Sagar Chaki, Edmund M. Clarke, Joel Ouaknine, Natasha Sharygina, and Nishant Sinha. State-event-based software model-checking. In *Integrated Formal Methods: 4th International Conference, IFM 2004, Canterbury, UK*, pages 128–147. Springer, April 2004.
- [Fis97] C. Fischer. Csp-oz: A combination of object-z and csp. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS'97)*, volume 2, pages 423–438. Chapman & Hall, 1997.
- [HDZ00] John Hatcliff, Matthew B. Dwyer, and Hongjun Zheng. Slicing software for model construction. *Higher-Order and Symbolic Computation*, 13(4):315–353, 2000.
- [HHF⁺94] J. He, C.A.R. Hoare, M. Franzle, M. Müller-Olm, E.-R. Olderog, M. Schenke, M.R. Hansen, A.P. Ravn, and H. Rischel. Provably correct systems. In *Formal Techniques in Real-Time and Fault Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 288–335. Springer, 1994.
- [HO02] J. Hoenicke and E.-R. Olderog. Csp-oz-dc: A combination of specification techniques for processes, data and time. *Nordic Journal of Computing*, 9(4):301–334, 2002.
- [Hoa78] C.A.R. Hoare. Communicating sequential processes. *CACM*, 21:666–677, 1978.
- [Hoa85] C.A.R. Hoare. *Communicating sequential processes*. Prentice Hall, 1985.
- [HRB90] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.*, 12(1):26–60, 1990.
- [HZ97] M.R. Hansen and C. Zhou. Duration calculus: Logical foundations. *Formal Aspects of Computing*, 9:283–330, 1997.
- [Lam83] L. Lamport. What good is temporal logic. *Information Processing*, 83:657–668, 1983.
- [Muc00] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 2000.
- [NNH99] Hanne Riis Nielson, Flemming Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer, 1999.
- [Smi00] G. Smith. *The Object-Z Specification Language*. Kluwer Academic Publisher, 2000.
- [Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall International Series in Computer Science, 2nd Edition, 1992.
- [Tip95] Frank Tip. A survey of program slicing techniques. *Journal of programming languages*, 3:121–189, 1995.
- [WD96] J. Woodcock and J. Davies. *Using Z – Specification, Refinement, and Proof*. Prentice-Hall, 1996.
- [Wei81] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.
- [ZHR91] C. Zhou, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.