
SEMINAR



FGI 3: Logik und Semantik von Programmen

Mo 12:15 - 13:45 in C-221

Rüdiger Valk

Ziel des Seminars



In dieser Veranstaltung werden Schlüsselqualifikationen durch

- * selbstständiges Recherchieren,
- * Strukturieren,
- * Präsentieren und
- * Moderieren erworben.

Die Zulassung zur Modulprüfung setzt die **regelmäßige Teilnahme** an dem Seminar, der Vortrag eines **Referats** und die Abgabe einer **Hausarbeit** voraus.

Hausarbeit

- * LaTex
- * mindestens 20, maximal 30 Seiten
- * Layout: Springer
- * Aufbau

LaTex

Modelling Concurrency with Quotient Monoids

Ryszard Janicki* Dai Tri Man Lê**

Department of Computing and Software,
McMaster University,
Hamilton, ON, L8S 4K1 Canada
{janicki,ledt}@mcmaster.ca

Abstract. Four quotient monoids over step sequences and one with compound generators are introduced and discussed. They all can be regarded as extensions (of various degrees) of Mazurkiewicz traces [14] and comtraces of [10].

Keywords: quotient monoids, traces, comtraces, step sequences, stratified partial orders, stratified order structures, canonical representations.

1 Introduction

Mazurkiewicz traces or partially commutative monoids [1, 5] are quotient monoids over sequences (or words). They have been used to model various aspects of concurrency theory since the late seventies and their theory is substantially developed [5]. As a language representation of partial orders, they can nicely model “true concurrency.”

For Mazurkiewicz traces, the basic monoid (whose elements are used in the equations that define the trace congruence) is just a free monoid of sequences. It is assumed that generators, i.e. elements of trace alphabet, have no visible internal structure, so they could be interpreted as just names, symbols, letters, etc. This can be a limitation, as when the generators have some internal structure, for example if they are sets, this internal structure may be used when defining the set of equations that generate the quotient monoid. In this paper we will assume that the monoid generators have some internal structure. We refer to such generators as ‘compound’, and we will use the properties of that internal structure to define an appropriate quotient congruence.

One of the limitations of traces and the partial orders they generate is that neither traces nor partial orders can model the “not later than” relationship [9]. If an event a is performed “not later than” an event b , and let the step $\{a, b\}$ model the simultaneous performance of a and b , then this “not later than” relationship can be modelled by the following set of two step sequences $s = \{\{a\}\{b\}, \{a, b\}\}$. But the set s cannot be represented by any trace. The problem is that the trace independency relation is symmetric, while the “not later than” relationship is not, in general, symmetric.

* Partially supported by NSERC grant of Canada.

** Partially supported by Ontario Graduate Scholarship.

Latex und den article style benutzen!

Frei verfügbare Versionen erhaltet Ihr z.B unter

<http://www.texshop.org>

<http://www.latex-project.org/>

<http://miktex.org/>

<http://www.tug.org/texlive/>

Empfehlung: Tetex (oder Livetex)

Enthalten in jeder Distribution oder für Nicht-Distributionen als integrierte Systeme erhältlich:

Windows: <http://www.cygwin.com>

Mac OSX: <http://fink.sf.net>

(Jeweils packages: tetex, (x)emacs, auctex)

Latex ist auch an der Uni installiert und sollte dort sofort laufen.

bibtex ermöglicht die Verwendung von separaten Literaturdateien.

(In dem Text selbst sind natürlich die entsprechenden Literatureinträge möglichst mit Seitenangaben (bei allem was länger als 20 Seiten ist) anzugeben.)

Nur Webseiten als Referenzen reichen nicht aus.

Unter

<http://www.springer.com/comp/lncs/authors.htm ...>

findet Ihr eine Vorlage für das Erstellen von
wissenschaftlichen Beiträgen mit Hilfe von Latex.

Achtung: <ftp://ftp.springer.de/pub/tex/latex/lncs/lat ...>
ist der direkte Zugriff auf die lncs-Version.

Beispiele für einen guten Aufbau findet Ihr unter den
tausenden von Artikeln, die Ihr bei Springer
findet!

Die Formatierungsvorlage liefert aber einen guten Rahmen
für das Layout, Schriftgröße, etc.

Insbesondere ergibt sich aber so eine Art Maß für den
Umfang der Ausarbeitung.

Bitte eine Rechtschreibprüfung verwenden.
Texte mit offensichtlichen Fehlern in Ausarbeitungen werden
zurückgegeben.

Tipp bei der Latex-Benutzung:
latex bibtex latex latex, erst dann sind auch die
Literatureinträge auch korrekt eingebunden.

Terminplan

* 13.10. Einführung und Themenvergabe

* Vortragstermine:

3. / 10. / 17. / 24. November

1. / 8. / 15. Dezember

5. / 12. / 19. / 26. Januar

* Es folgen die Themenvorschläge

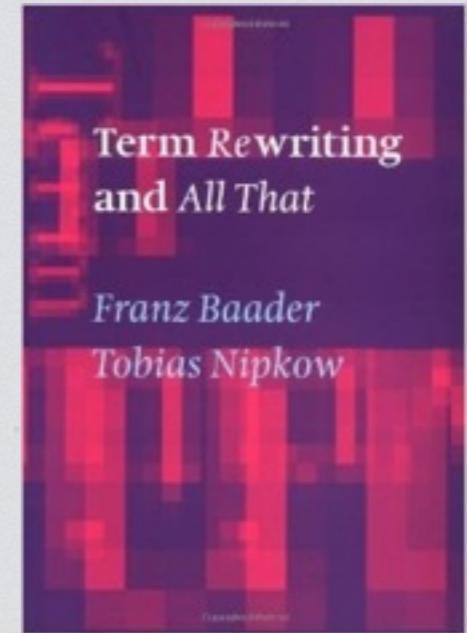
*

I. Term-Ersetzungs-Systeme und ihre Termination

3.11.



@book{BaNi98,
Author = {F. Baader and T. Nipkow},
Publisher = {Cambridge University Press},
Title = {Term Rewriting and All That},
Year = {1998}}



Skript:

[http://www.informatik.uni-hamburg.de/TGI/lehre/vl/WS0708/FGI3/sec/Semantik_Kap5_1\(ersetzung\).pdf](http://www.informatik.uni-hamburg.de/TGI/lehre/vl/WS0708/FGI3/sec/Semantik_Kap5_1(ersetzung).pdf)

Benutzer: tgiskript
PW: Kneissl

Definition 5.2 Einer Reduktion \rightarrow werden folgende Eigenschaften zugeschrieben:

a) Church-Rosser, falls $x \xrightarrow{*} y \Rightarrow x \downarrow y$



grünes Beispiel:

$$A = \{2, 3, 4, \dots\}$$

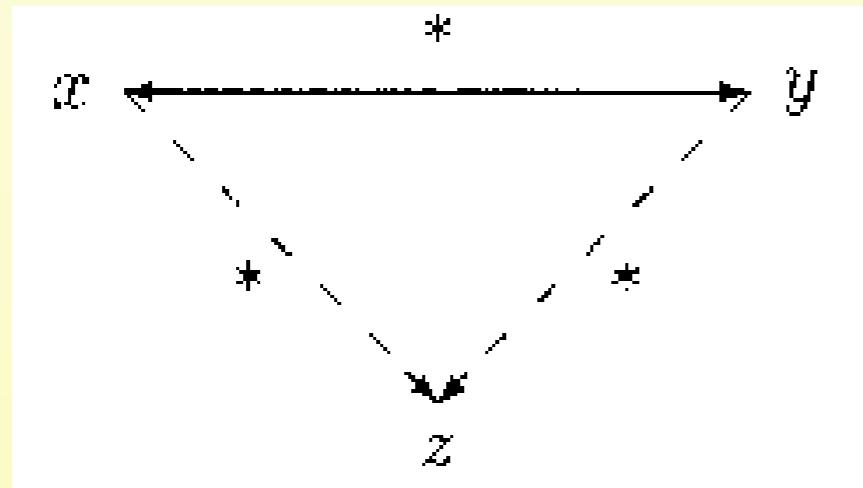
$$x \rightarrow y : \Leftrightarrow x > y \wedge y | x$$

$$\xrightarrow{*} = A \times A$$

nicht Church-Rosser

Definition 5.2 Einer Reduktion → werden folgende Eigenschaften zugeschrieben:

a) Church-Rosser, falls $x \xleftrightarrow{*} y \Rightarrow x \downarrow y$



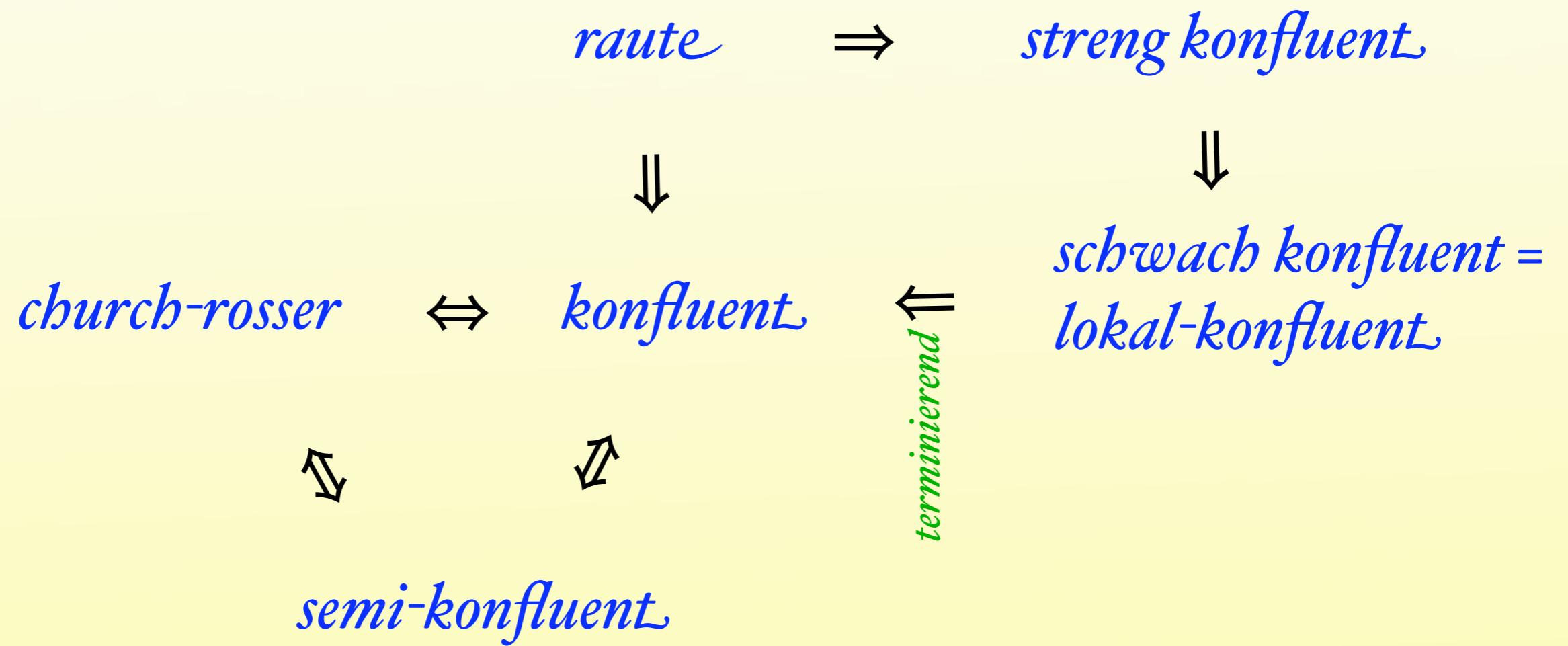
grünes Beispiel:

$$A = \{2, 3, 4, \dots\}$$

$$x \rightarrow y : \Leftrightarrow x > y \wedge y|x$$

$$\xleftrightarrow{*} = A \times A$$

nicht Church-Rosser



3 Axiomatische Semantik

10.11.

Floyd, R.W.: Assigning meanings to programs. Mathematical Aspects of Computer Science, XIX American Mathematical Society (1967).

Hoare, C.A.R.: An axiomatic approach to computer programming,
Comm. ACM 12 (1969), 576-580, 583

Apt, K.R., Olderog, E.R.: Programm-Verifikation—Sequentielle, parallele und verteilte Programme, Springer Verlag, Berlin, 1994
(auch: Verification of Sequential and Concurrent Programs, Springer Verlag, Berlin 1991).

Skript:

[http://www.informatik.uni-hamburg.de/TGI/lehre/vl/WS0708/FGI3/sec/Semantik_Kap5_1\(ersetzung\).pdf](http://www.informatik.uni-hamburg.de/TGI/lehre/vl/WS0708/FGI3/sec/Semantik_Kap5_1(ersetzung).pdf)

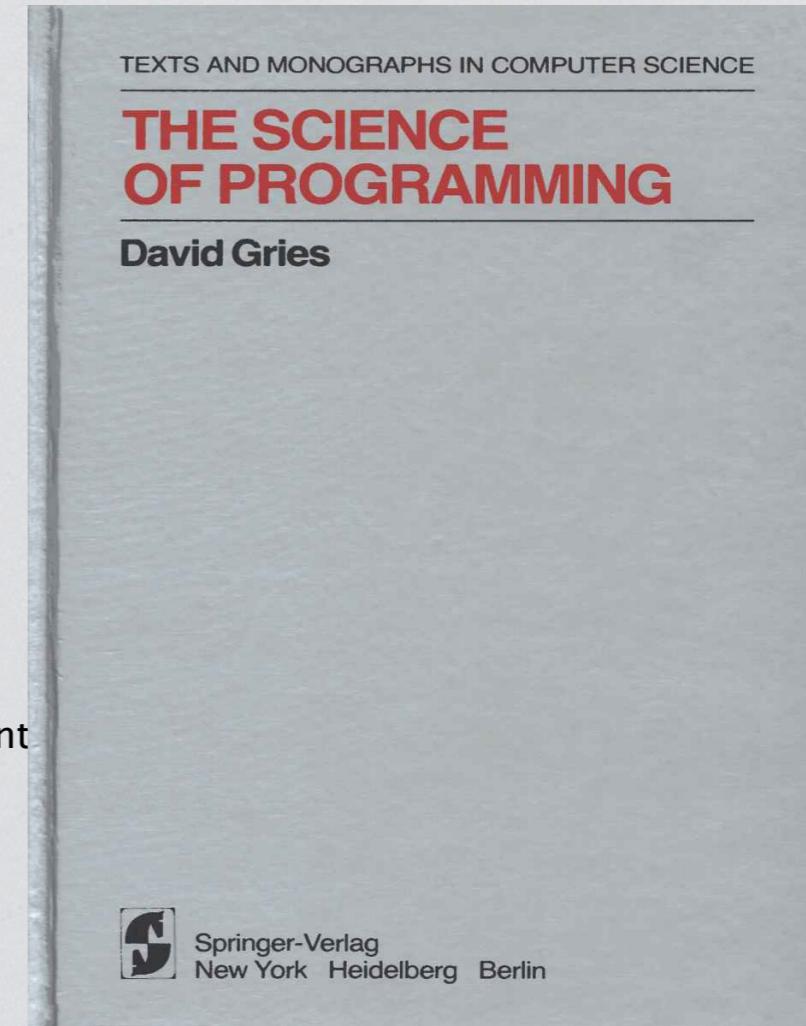
Benutzer: tgiskript
PW: Kneissl

4.

Programmieren mit und Entwicklung von Invarianten

17.11.

D. Gries: The Science of Programming (Springer, 1981) [T GRI]



Skript:

<http://www.informatik.uni-hamburg.de/TGI/lehre/vl/WS0708/FGI3/sec/Semant>

Benutzer: tgiskript

PW: Kneissl

5. Verteilte Algorithmen: Verteilte Termination

24.11.

F. Mattern: Verteilte Basisalgorithmen,
Springer, Berlin, 1989

4 Verteilte Terminierung

Friedemann Mattern

Verteilte
Basisalgorithmen



Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo Hong Kong

Das auf den ersten Blick einfach erscheinende Problem festzustellen, ob eine verteilt ablaufende Berechnung beendet ist, ist überraschenderweise nicht trivial. Es handelt sich dabei um eine Fragestellung, die erst bei verteilten Systemen auftritt und im sequentiellen Fall überhaupt nicht existiert: Bei sequentiellen Programmen ist zwar i.a. auch von vornherein nicht entscheidbar, ob ein Programm terminiert – wenn zur Laufzeit das Programmende erreicht wird, so kann diese Tatsache jedoch problemlos als solche erkannt und nach "außen" gemeldet werden; es ist also einfach feststellbar, daß ein sequentielles Programm beendet ist. Genau dies ist bei verteilten Programmen schwierig, da temporär terminierte Prozesse durch eintreffende Nachrichten, die von noch aktiven Prozessen ausgesendet wurden, wieder reaktiviert werden können.

Das sogenannte Problem der *verteilten Terminierung* läßt sich als ein spezielles Schnapschüßproblem auffassen, bei dem der feststellende globale Zustand die Eigenschaft hat, daß alle Prozesse terminiert sind und keine Nachrichten mehr unterwegs sind. Offensichtlich ist die so definierte globale Terminierung ein stabiles Prädikat des Systemzustands. Das erstmalig um 1980 unabhängig von Francez [FRA79a, FRA80a] sowie Dijkstra und Scholten [DIS80a] beschriebene Problem stellt einen der am besten untersuchten Aspekte verteilter Berechnungen dar, alleine bis zum Jahr 1987 sind über 50 Lösungsvorschläge veröffentlicht worden (vgl. dazu die Literaturreferenzen in [MAT87a]; in jüngster Zeit ist noch eine ganze Reihe weiterer Veröffentlichungen mit einer überraschenden Vielzahl z.T. sehr unterschiedlicher Lösungsalgorithmen (im folgenden oft einfach verkürzt mit "Terminierungsalgorithmen" bezeichnet) hinzugekommen.

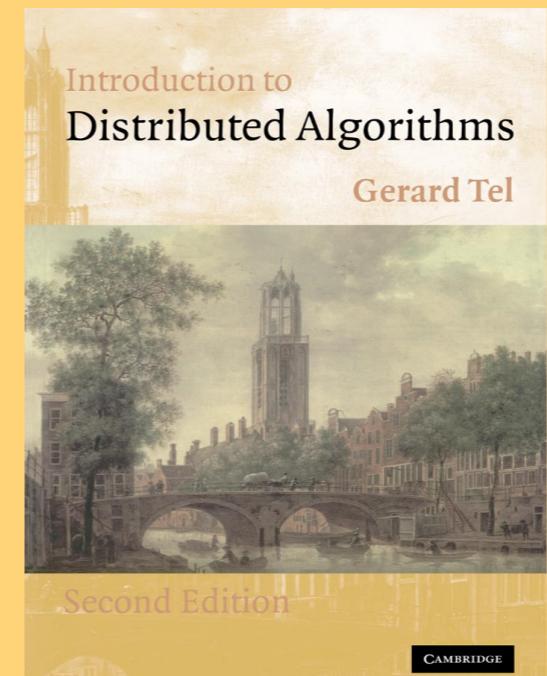
4.1 Einleitung

Das Problem der verteilten Terminierung kann zunächst informell so formuliert werden:

Wie kann in einem verteilten System *festgestellt* werden, daß eine verteilte Berechnung beendet ist?

Da jeder Prozeß zunächst nur für sich selbst entscheiden kann, ob er terminiert ist (d.h. höchstens noch aufgrund einer empfangenen Nachricht wieder aktiv wird), jedoch keine konsistente globale Sicht auf das Gesamtsystem hat, ist das Problem keineswegs

G. Tel: Introduction to Distributed Algorithms



8

Termination Detection

A computation of a distributed algorithm terminates when the algorithm has reached a *terminal configuration*; that is, a configuration in which no further steps of the algorithm are applicable. It is not always the case that in a terminal configuration each process is in a *terminal state*; that is, a process state in which there is no event of the process applicable. Consider a configuration where each process is in a state that allows the receipt of messages, and all channels are empty. Such a configuration is terminal, but the processes' states could also occur as intermediate states in the computation. In this case, the processes are not aware that the computation has terminated; and the termination of the computation is said to be *implicit*. Termination is said to be *explicit* in a process if the state of that process in the terminal configuration is a terminal state of the process. Implicit termination of the computation is also called *message termination* because no more messages are exchanged when a terminal configuration has been reached. Explicit termination is also called *process termination* because the processes have terminated if an algorithm explicitly terminates.

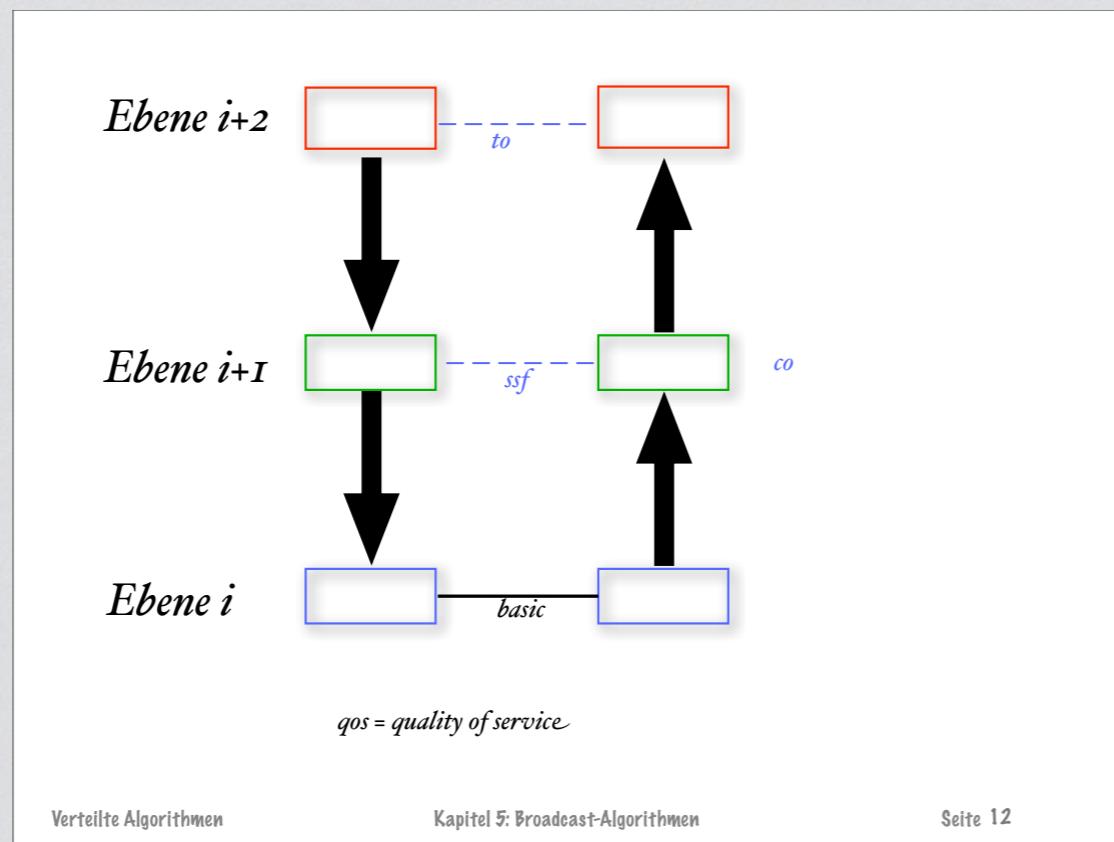
It is usually easier to design an algorithm that terminates implicitly than one that terminates explicitly. Indeed, during the design of an algorithm all aspects regarding the proper termination of processes can be ignored; the design concentrates on bounding the overall number of events that can take place. On the other hand, the application of an algorithm may require that processes terminate explicitly. Only after explicit termination can the results of a computation be regarded as final, and variables used in the computation discarded. Also, a *deadlock* of a distributed algorithm results in a terminal configuration; in this case the computation must be restarted when a terminal configuration is reached.

In this chapter general methods will be investigated for transforming

6. Verteilte Algorithmen: Broadcast-Protokolle

1.12.

H. Attiya, J. Welch: Distributed Computing, McGraw-Hill, London, 1998



Distributed Computing

*Fundamentals, Simulations
and Advanced Topics*

Second Edition

Hagit Attiya
Jennifer Welch



A JOHN WILEY & SONS, INC., PUBLICATION

Skript: Abschnitt 8.5 aus

[http://www.informatik.uni-hamburg.de/TGI/lehre/vl/SS11/VAlg/sec/Kap5-VA\(broadcast\)-skript.pdf](http://www.informatik.uni-hamburg.de/TGI/lehre/vl/SS11/VAlg/sec/Kap5-VA(broadcast)-skript.pdf)

7. Das Residuum von Petrinetzen

8.12.

The Residue of Vector Sets with Applications to Decidability Problems in Petri Nets

Rüdiger Valk and Matthias Jantzen

Fachbereich Informatik (TGI), Universität Hamburg, Rothenbaumchaussee 67,
D-2000 Hamburg 13,
Bundesrepublik Deutschland

Nets with Tokens Which Carry Data

Ranko Lazić^{1,*}, Tom Newcomb², Joël Ouaknine²,
A.W. Roscoe², and James Worrell²

¹ Department of Computer Science, University of Warwick, UK

² Computing Laboratory, University of Oxford, UK

Computing Minimal Elements of Upward-Closed Sets for Petri Nets

Hsu-Chun Yen* and Chien-Liang Chen

Dept. of Electrical Engineering, National Taiwan University

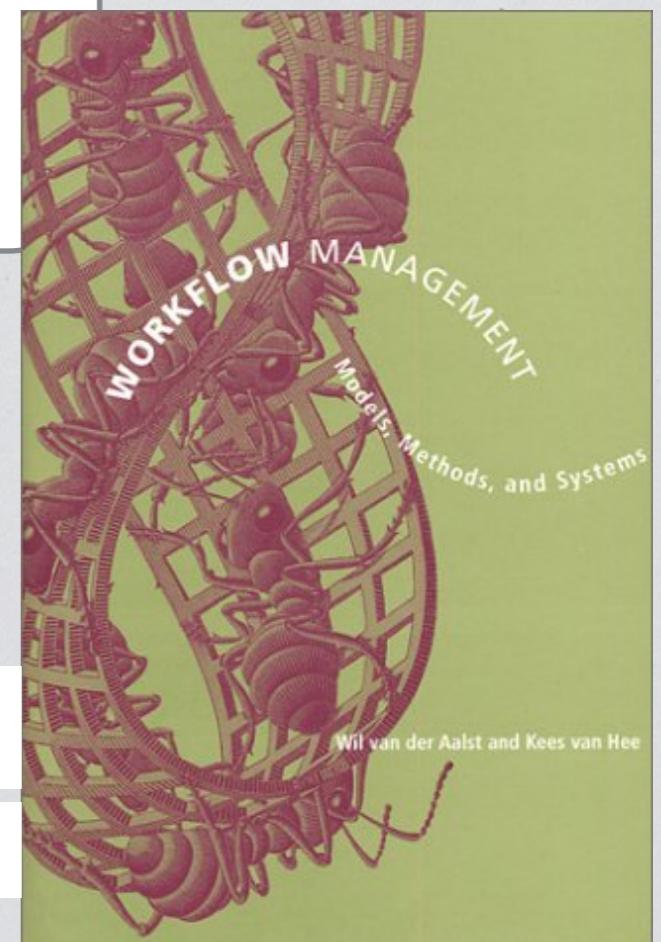
Taipei, Taiwan 106, Republic of China

yen@cc.ee.ntu.edu.tw

8. Workflow-Petrinetze

15.12.

- [Aal97] W.M.P. van der Aalst. Verification of Workflow Nets. volume 1248 of *Lecture Notes in Computer Science*, pages 407–426, Toulouse, France, 1997. Springer-Verlag.
- [Aal98] W.v.d. Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8:21–66, 1998.
- [Aal00] W.v.d. Aalst. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In W.v.d. Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management*, volume 1806 of *LNCS*, pages 161–183. Springer-Verlag, Berlin, 2000.



4

Analyzing Workflows

9. Erweiterungen von Petrinetzen

5.1.

Petri Nets with Marking-Dependent Arc Cardinality: Properties and Analysis

Gianfranco Ciardo

24. R. Valk. On the computational power of extended Petri nets. In *Seventh Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 64*, pages 527–535. Springer-Verlag, 1978.
25. R. Valk. Generalizations of Petri nets. In *Mathematical foundations of computer science, Lecture Notes in Computer Science 118*, pages 140–155. Springer-Verlag, 1981.

10. Komplexität eingeschränkter Objekt-Netze

12.1.

**Structural and Dynamic Restrictions of
Elementary Object Systems**

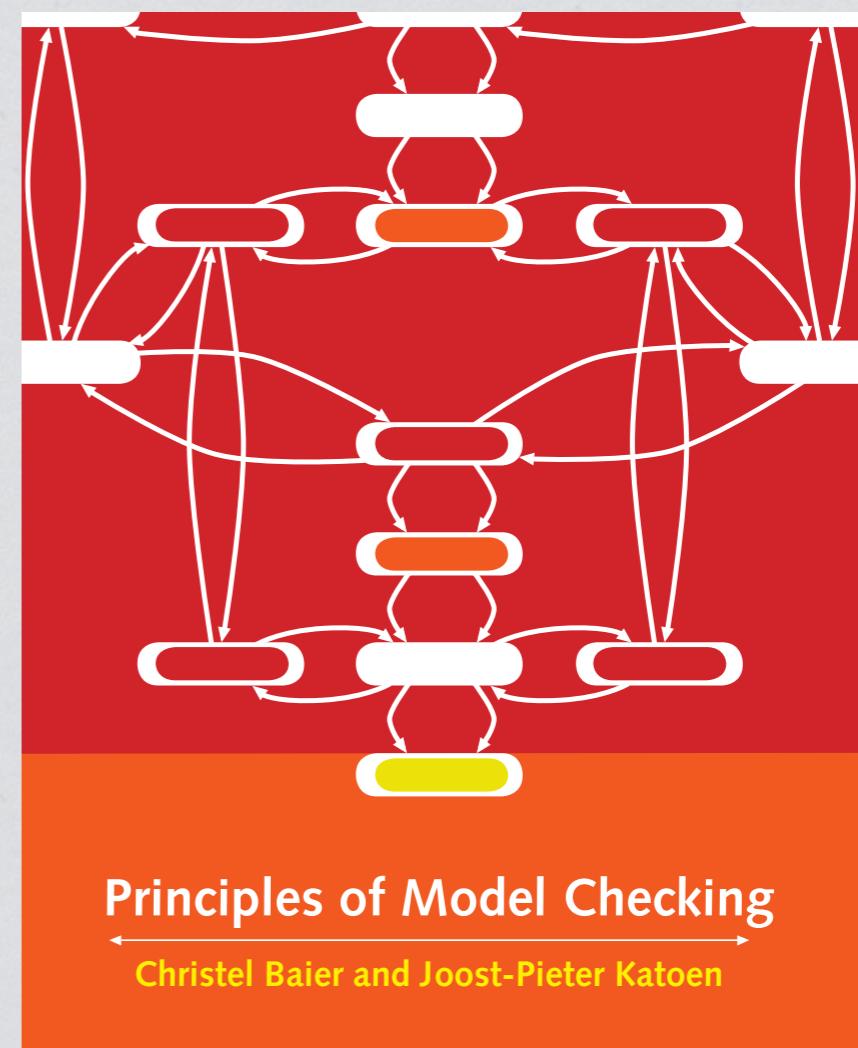
Frank Heitmann^C and Michael Köhler-Bußmeier

**Algorithms and Hardness Results
for
Object Nets**

II. Temporale Logik und Bisimulation

19.I.

@book{Baier_Katoen,
Author = {C. Baier and J.-P. Katoen},
Date-Added = {2008-08-26 13:25:38 +0200},
Date-Modified = {2008-08-26 13:33:09 +0200},
Keywords = {Model Checking},
Publisher = {MIT press},
Title = {Principles of Model Checking},
Year = {2008}}



12. Sicherheits- und Lebendigkeits-Eigenschaften als ω -Sprachen

26. I.

@book{Baier_Katoen,
Author = {C. Baier and J.-P. Katoen},
Date-Added = {2008-08-26 13:25:38 +0200},
Date-Modified = {2008-08-26 13:33:09 +0200},
Keywords = {Model Checking},
Publisher = {MIT press},
Title = {Principles of Model Checking},
Year = {2008}}

