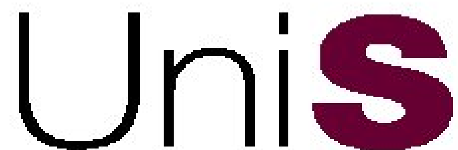
Bibliography and References

Graham G. Thomason

Appendix to the Thesis “The Design and Construction of a State Machine System that Handles Nondeterminism”



Department of Computing
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

July 2004

© Graham G. Thomason 2003-2004

Summary

This annotated bibliography accompanies the thesis on *The Design and Construction of a State Machine System that Handles Nondeterminism* (called STATECRUNCHER) and is divided into five parts: (1) internal Philips publications relating to (conformance) testing, setting a backdrop; (2) systems and formalisms supporting state machines; (3) publications relating to state machines; (4) supporting projects / products / information of relevance to testing; (5) a consistent set of STATECRUNCHER references. In addition to state-based techniques, various other model-based testing techniques are touched upon within the various categories.

 © Graham G. Thomason 2003-2004

Table of Contents

1. Introduction 1
   1.1 Categorisation of references 1
   1.2 Abbreviations and definitions used in this appendix 2
2. Internal Philips publications 3
3. Systems and formalisms supporting state machines or related models 8
4. Publications relating to verification, testing and/or state machines 19
5. Supporting projects / products / information 38
6. STATECRUNCHER references 44

© Graham G. Thomason 2003-2004

1. Introduction

1.1 Categorisation of references

The references have been arranged in categories, then alphabetically, as follows

- Internal Philips publications relevant to validation and verification (testing)
- Systems and formalisms supporting state machines and other model-based testing techniques
- Publications relating to state machines and other model-based testing techniques
- Supporting projects/products/information of relevance to testing
- The STATECRUNCHER references.

The Philips reports show some of the history in the company of state-based conformance testing, as a backdrop to the development of STATECRUNCHER.

Under *systems supporting state machines*, we include *model checking systems*, because whether or not they offer a simulation facility, they internally run some state machine engine. We will distinguish two kinds of tool in our annotations (rather than introducing separate categories): *model checkers* and *simulators/test oracles*. The corresponding activities may be called *validation* and *verification/testing* respectively, though *'verification'* is often used of model checking, and we often meet the phrase 'verifying properties'. A software system needs a *design* and an *implementation*, and both need a separate kind of tool and activity to ensure the quality of the final system.

- The design must guarantee certain properties, e.g. safety, liveness, fairness, freedom from deadlock. Given a formal design, such as a statechart with properties attached to states, and a formulation of the properties required in a system, a model checker can attempt to prove them. Two possible limitations are: the expressiveness of the property language (typically a temporal logic), and the size of the state space (though some techniques allow for vast numbers of states).
- Given a design, the system must be built. Televisions, mobile phones etc. are a combination of hardware and software. The concept of *being in a state* means much more to a real system than to a simulator: mobile phone transmitters may be switched on, threads may be waiting for semaphores, buffers should have certain content, such as a teletext page. Testing involves making sure that these things that should happen really do happen. The state model tells us what it is that should happen.

A slogan popular in Philips in the 1990s was: ***Doing the right thing and doing things right***. This is like saying: ***validating*** the properties of the design, and ***verifying*** (testing) that the implementation conforms to the design. Both are extremely important, but distinct, though an occasional tool (e.g. SPIN) is suited to both.

We also note in our annotations whether a state-based testing system is of the ***Labelled Transition System*** (LTS) type or *(Mealy)* ***Finite State Machine*** (FSM) type. The former has

affinities with CCS and CSP; event sequences are the traces, and events are partitioned into input events and output events. The FSM approach defines a separate output alphabet. FSMs produce output on transitions, the *trace* of such systems. [Tretmans] regards the precise relation between testing theories based on the two approaches as an aspect of further study. STATECRUNCHER was designed as a *test oracle*, and the main thrust of the thesis is that its design will help in ***testing***. Nevertheless it could be used to validate properties, given the aid of an additional tool communicating with it, because it offers facilities which can help in exploring state spaces. STATECRUNCHER is more geared to the FSM approach than the LTS approach. In [StCrSemComp] we make some comparisons with the process algebras. Some papers describe work where the implementation language is SDL; this corresponds more to an LTS approach than an FSM one, because input and output messages are both analogous to events.

The main scope of the bibliography is *state-machine systems* (and how they have been used), whether commercial, proprietary, or academic, principally in a testing context, but also in a validating context. *Test generation algorithms* are surveyed, as being STATECRUNCHER's nearest neighbour in a tool chain. In addition we give some references for UML-based modelling other than dynamic modelling, and we mention a few other testing techniques: cause effect graphing, orthogonal array testing.

Under *supporting projects/products/information* we cover various tools, which, although they may appear to be a disparate collection, have proved to be of especial value in constructing testing tools and synthesizing tool chains. PROLOG features prominently in the list, being the implementation language of STATECRUNCHER.

Finally, the STATECRUNCHER references form a consistent set of documents describing the system from various angles at its latest release (1.05).

1.2 Abbreviations and definitions used in this appendix

We use abbreviations and technological terms, where not explained, sparingly in the annotations, but the following are so commonly needed as to be useful:

Black box	Used of a state machine, this means that states themselves are not directly observable, but outputs on transitions are, and it is from these that a state may be deduced.
FSM	Finite State Machine
IUT	Implementation Under Test
LTS	Labelled Transition System
NFSM	Nondeterministic Finite State Machine
OSI	Open Systems Interconnection
SUT	System Under Test

© Graham G. Thomason 2003-2004

2. Internal Philips publications

The Philips laboratories involved are:

- PRL (Philips Research Laboratories, Redhill)
- PDSL-R (Philips Digital Systems Laboratories, Redhill)
- Nat. Lab. (*Natuurkundig Laboratorium*, Philips Research Laboratories, Eindhoven)
- PRI-B (Philips Research India - Bangalore).

These reports cover state-based testing and related issues in various ways: early studies, tooling approaches, transition tour approaches, and case studies.

[BakerM] **M.L. Baker and D.C. Yule**
Automation of Software Testing:
A Case Study on a Real-Time Embedded System
PRL Technical Note 3373, September 1995

This report describes early work within Philips Research to automate testing of two Interactive TV applications (an interactive quiz show and interactive shopping –both teletext based). The work featured:

- state-based testing, using the public domain tool [DejaGnu] as a test harness, with custom code being written in Expect/TCL. The state behaviour was defined using state-relation tables.
- code coverage, using the [McCabe] toolset.

Out of 1400 tests, 76 failed. Two major errors relate to a requirements omission and an implementation omission. The combination of the two techniques makes it possible to see how much code is exercised by a state model. Branch coverage (stronger than statement coverage) figures in modules varied from 26% to 100%. The low figures were often where error recovery code had not been exercised; more tests could be devised to increase the coverage.

[ECHSM] **M.J. Hollenberg**
Extended Hierarchical Concurrent State Machines,
Syntax and Semantics
Nat. Lab. Report, version 0.4, 25 October, 1999

This is a document describing the syntax for an ECHSM (Extended Concurrent Hierarchical finite State Machine) language. The syntax is an extension to that of [CHSM]. The semantics are practically “as in [CHSM]”. The purpose of the language

is to flatten ECHSM's to FCHSM's (see [FCHSM]) for use with [PHACT]. The grammar has been largely adopted by STATECRUNCHER, with extensions, and with the semantics extended for nondeterminism.

[FCHSM] **M.J. Hollenberg**
Flattened Concurrent State Machines, Syntax and Semantics
Nat. Lab. Report, version 0.2, October 25, 1999.

A language for describing flattened concurrent hierarchical state machines, derived from ECHSM's (see [ECHSM]), for use with [PHACT].

[GFET] **G.G. Thomason**
A GUI Front End for Testing
Program GFET (Multi-threaded version)
User Manual, Version 2.0/5.0
PRL Technical Note 3875, July 1999

A tool to give a Windows user interface to embedded software that does not have a user interface. It allows for control of 10 threads on which portions of software can be run. It provides easy implementation of stubbed functions as dialogue boxes. This enables the software to be tested using button-pressing, edit-box-communicating Windows software testing tools, such as WinRunner [WinRun] to test embedded software. The test script may make use of a state-relation package [Trew 98].

[Koppalkar 02] **Nitin Koppalkar and Animesh Bhowmick**
Integration of Generic Explorer with the TorX Tool Chain
Nat. Lab. Technical Note 2002/387, October 2002.

This report describes how STATECRUNCHER, being an explorer in [TorX] terms, can be integrated into the TorX tool chain. The actual integration took place later, when STATECRUNCHER had a socket interface.

[Koppalkar 03] **Nitin Koppalkar**
Interfacing STATECRUNCHER with TorX for demonstrating the state-based testing technique taking MG-R components for a case study
Nat. Lab. Draft Report, December 2003

This report shows STATECRUNCHER in the [TorX] tool chain in action testing a TV software component.

 © Graham G. Thomason 2003-2004

[Koymans] **Ron Koymans**
An Overview of Automatic Test Generation Techniques for Communication Protocols
Nat. Lab. Report RWR-508-re-93558, November, 1994

The report describes the relevant state of the art (at the time of writing) in conformance testing, with explanations of test sequence generation by the T, D, W and U methods: transition tours, distinguishing sequences, characterisation sets and unique I/O sequences, and extensions to these. Tooling is SDL, LOTOS and Estelle based, with TTCN used as a test definition format.

[Lanaspre] **B. Lanaspre**
A Statechart Pre-processor for an Automatic Test Case Generator
PRL Technical Note 3912

This report describes how a state-based model written in [CHSM] can be flattened, and then have its variables expanded, to give final output in a Flattened State Machine language to be used as input to [PHACT]. The flattening process takes place by driving CHSM through its state space. The concepts were used in testing American digital television (DTV '98).

[PHACT] **L. Heerink and M.J. Hollenberg**
Conformance Testing Using PHACT
Nat. Lab. Technical Note NL-TN 2000/011 (5 Jan 2000)

PHACT (Philips Automated Conformance Tester) is built on a proprietary state-based testing tool, KPN's *Conformance Kit*. KPN [http://www.kpn.com] is a large Dutch telecom company, the main successor to the Dutch PTT. PHACT does not support hierarchy (so hierarchical state models must be flattened). It has been used to test an MPEG source decoder (DIVA5) and American digital TV (DTV'98). Some handling of nondeterministic situations can be managed by defining intermediate states [p.41].

[Raptis 98] **D. Raptis**
Generation of Test Sequences from FSM's
PRL Technical Note 3683, March 1998

The problem addressed in this report is that of generating *transition tours* round a state transition diagram. A tour is then effectively a black-box test sequence, since it does not rely on being able to set any state directly, (which would be white-box control). The problem of generating the tour is known as the Chinese Postman Problem. Part of the solution is to solve an *assignment problem*. For an optimal solution, Raptis refers us to the *Hungarian* solution, *Christos H. Papadimitriou and Kennett Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, 1982*. This has cubic complexity. Raptis presents a faster algorithm for a non-optimal, but near-optimal solution, with some experimental results.

© Graham G. Thomason 2003-2004

[Raptis 99a] **D. Raptis**
A Modelling and Testing Approach for Horizontal Communication in the TV Platform
PRL Technical Note 3893, April, 1999

This report describes how [CWB] (Concurrency Workbench) was used to model the state-based behaviour of the *composition* of two formal software components given their interface specifications. The components handle parts of an end-to-end analogue signal flow: a tuner and high-end output processor. The interactions of such components are only with adjacent components (horizontal communication) - so obviating the need for a manager program that knows the whole configuration. This scheme facilitates system synthesis from components, but integration testing is needed to ensure it works.

[Raptis 99b] **D. Raptis**
Modelling and Validation of Concurrent Programs using CCS
PRL Technical Note 3896, August, 1999.

This report shows how CCS agents, with and without value passing, can be designed to model data types, variables and algorithms. Semaphores and Peterson's algorithm for mutual exclusion are described as examples. A pre-processor using a Unix *sed* script is described for translating from a user-friendly syntax to CCS. An introduction to verification of model properties as supported by CTL*, rather than the modal mu calculus of CCS, is given.

[Thomason] **G.G. Thomason**
Component Binding in Composite Models for State-based Testing
PRL Technical Note TN 4102, August, 2001

The aim of this report is to identify how systems built from software components will need to be tested. A tool chain is required which can automatically *generate* and *execute* tests —in particular integration tests. The generation side must use models of the behaviour of individual components and of their binding which 'wires up' the complete system, and produces tests and their 'oracle' from the model —which may incorporate several alternative results in the event of nondeterminism. Solutions are explored involving compositions of STATECRUNCHER models, using a preprocessor to make model bindings in the same way that system bindings are made.

[Trew 98] **T.I.P. Trew**
State-based Testing with WinRunner: the State-Relation Package
PRL Internal Note SEA/704/98/05, June 1998

This package, allows a WinRunner [WinRun] test script to loop over tests defined by state relation tables and so execute state-based tests.

 © Graham G. Thomason 2003-2004

[Trew 01] **T.I.P. Trew**

Software Component Composition - Still "Plug and Pray?"

Proceedings of the 6th Philips Software Conference, February, 2001

This presentation describes the difference between ordinary object-oriented development and component development and the impact of that on testing. The need for good, structured integration testing is all the more important. (State based testing can be expected to be a major part of this).

[Trew 03] **T.I.P. Trew**

State-based modelling of software components for integration testing

A practical guide to the creation of STATECRUNCHER models

Philips Nat. Lab. Technical Note ***(under preparation).***

This report addresses the practicalities of using STATECRUNCHER to model systems of software components.

[VnV] **Eleen Hollenberg and Erik Mallens**

CvnvTestframe User Manual

MG-R Software Documentation, v2.0, October 2001.

This is a Philips proprietary test harness for embedded systems with a host side part and a target side part.

[Yule] **D.C. Yule**

Automatic State-Based Testing ***(of various modules)***

PRL Technical Notes TN 3574 / 3681 / 3582 / 3590, 1997

or **DVD Document V19 C4 S415.**

This illustrates the effectiveness of state-based testing. In a DVD player, errors (sometimes many) were found in *every* module tested – even though this was after hand-crafted conventional tests had been run. The modules were: the Loader Subsystem, the Media Access module, the CD-DA Playback module, and the VCD Playback module.

© Graham G. Thomason 2003-2004

3. Systems and formalisms supporting state machines or related models

[Agedis] **www.agedis.de**

A consortium project headed by IBM Research Laboratory, Haifa, with the aim of "...automating software testing and improving the quality of software while reducing the expense of the testing phase... by developing a methodology and tools for the automation of software testing in general, with emphasis on distributed, component-based software systems". A publication *Model based test generation tools* by Alan Hartman gives a list of the main tools available. Commercial tools: [TVEC], [Conformiq], [Reactis], Jcontract, [Tau], Testmaster, Unitek. Proprietary tools: [GOTCHA-TCBeans], Ucbt-Salt, [ASML], [PTK]. Academic tools: Spectest, Mulsaw, Toster, TGV/CADP, [TorX]/CADP, [Cow_Suite].

[Argos] **F. Maraninchi**

The Argos Language: Graphical Representation of Automata and Description of Reactive Systems

IEEE Workshop on Visual Languages, Kobe, Japan, October 1991

Argos supports the graphical development of statecharts. The graphical items correspond to a syntax, which directs the graphical editor. Nondeterminism is detected so that the user can remove it. The system supports UML-like models, including (synchronous) broadcast events. Verification is performed in an environment called Argonaute, using an automaton comparator called Aldebaran, for which the following reference is given: J.C. Fernandez, *An Implementation of an Efficient Algorithm for Bisimulation Equivalence*, Science of Computer Programming, vol. 13, 2-3, May, 1990. That article and additional information on Aldebaran can be found on the internet at the INRIA (Institut National de Recherche en Informatique et en Automatique) site: http://www.inrialpes.fr

[ARTISAN] **http://www.artisansw.com/**

http://www.artisansw.com/products/professional_overview.asp

From the Real Time Studio Professional web page

> "Already an acknowledged leader in providing modelling support for system engineers, ARTiSAN has added a powerful set of new enhancements to its system validation functionality, so that engineers can:
>
> - Build and simulate advanced state models for system behaviour:

© Graham G. Thomason 2003-2004

- Use events straight from the system architecture model
- Add timers and timed events
- Use drag/drop to populate state triggers, actions and guards
- Verify system response to external and internal events before building"

The transition semantics appear to be in agreement with UML.

[ASML] **(Abstract State Machine Language)**
http://research.microsoft.com/fse/asml

The above site includes a 76-page tutorial for download. ASML is "an executable specification language based on the theory of Abstract State Machines....good for testers...". The language is very reminiscent of *imperative languages*, (such as 'C++' – ASML has *classes*), rather than the *reactive systems* approach of other state machine systems such as [STATEMATE]. It has processing blocks divided into *steps,* allowing parallelism within steps, where updated variable values only take effect after a step. The notion of state is simply related to variable values at the end of a step, and transitions are the act of processing a step. The language includes sets and sequences, and maps (equivalent to associative arrays of Perl, or hash tables in database systems) Nondeterminism can be specified, but the system then makes one choice. There is support for predicate logic, e.g. `forall`...`holds` and `exists`...`where`. Microsoft state that ASML is being used for conformance checking. For a paper on Sequential Abstract State Machines, see [Gurevich].

[Caliber] **http://www.nohau.se/products/kravhantering.html**

A cause-effect graphing tool that has been used at Philips, originally called SoftTest from Bender and Associates, then apparently under Borland called Caliber-RBT and now under Nohau called Caliber-RM. Cause-effect graphs are described in [Myers].

[CCS] **Calculus of Communicating Systems**

A process calculus. See [Milner], [Bruns]

[CHSM] **Paul J. Lucas**
An Object-Oriented System for Implementing Concurrent, Hierarchical, Finite State Machines
MSc. Thesis, University of Illinois at Urbana-Champaign, 1993
http://homepage.mac.com/pauljlucas/software.html

CHSM stands for Concurrent Hierarchical finite State Machines, and (in context) Lucas's implementation of a language for them. The concurrency and hierarchy are expressed as 'sets' and 'clusters'. It allows for transition actions, which may be broadcast (i.e. fired) events. The language is easy to grasp, and although apparently not designed with testing applications in mind, it is at a suitable level for ordinary developers and testers to use. The language is implemented by conversion to C++ using the Unix tools YACC and LEX. CHSM supports embedded C++ in a source

model. CHSM prevents transition ‘cycling’ (potentially possible through broadcast events) by only allowing any one transition to be taken once during the processing of a top-level event. A CHSM model may contain nondeterministic transitions, but the system will take just the first one it finds.

[Conformiq] **http://www.conformiq.com**

A commercial tool supporting batch and on-the-fly testing, based on UML dynamic models.

[Cow Suite] **Francesca Basanieri, Antonio Bertolino, Eda Marchetti**
The Cow_Suite Approach to Planning and deriving Test Suites in UML Projects
Instituto di Elaborazione della Informazione, Pisa

Cow_suite tools generate test cases from UML diagrams, based on the analysis of Use Case diagrams and Sequence Diagrams. No translation into an intermediate notation is needed. A cost-weighted strategy is used, assigning weights to nodes of derived trees, to select the most ‘important’ test cases from all possible use cases and message sequences. The user can choose either a fixed number of tests, or fixed functional coverage. Managers provide ‘importance’ criteria. Cow_suite does not execute tests; for this a separate driver is required.

[CSP] **Communicating Sequential Processes**

A process calculus. See [Hoare], [Schneider].

[CTL] **Computation Tree Logic.**

This temporal logic is embodied in a language called CTL*. See [Emerson], [Bérard].

[CWB] **The Edinburgh Concurrency Workbench**
http://www.dcs.ed.ac.uk/home/cwb/

This tool expresses its designs in the Calculus of Communicating Systems (CCS). It is a powerful tool, and is popular as a research tool, but it is not aimed at the ordinary software developer in industry. It supports nondeterminism at a *transition* level, so that the user can choose between transitions even where some of them are triggered off the same event. (Contrast this with STATECRUNCHER, which supports nondeterminism at the *event* level, relieving the user of the need to detect and manage multiple nondeterministic transitions in their own loop).

[Design/CPN] **Design/Coloured Petri Nets**

Initially developed by Meta Software Corp, Cambridge MA USA, and the CPN Group at the University of Århus, Denmark. Available from

http://www.daimi.au.dk/designCPN

 © Graham G. Thomason 2003-2004

Design/CPN allows one to edit, simulate and verify large hierarchical coloured Petri nets ([Bérard, Ch14]). Since Petri nets can be used to model state-based systems (see [Murata]), the tool can be used to verify them.

[ESPRESS] **Engineering of safety-critical embedded systems**

http://www.first.gmd.de/~espress

"ESPRESS aims to increase productivity in developing complex, safety-critical, embedded systems and enhance the reliability of such systems by the development of a methodological tool-supported software technology for specific application areas covering the whole life-cycle. The project focuses on the application area of automobile electronics and traffic light control. ... Essential features are the explicit separation of specifications into functional and safety relevant parts, the combination of graphical (statecharts) and formal methods (Z) as well as verification, code-generation, systematic testing and automatic test evaluation."

Tool support is based on STATEMATE. See [Büssow] for a description of the formalism used: **μSZ.** See [Fuhrmann] for another ESPRESS publication, on the verification of STATEMATE statecharts via the CSP verification tool [FDR].

[Estelle] **ISO 9074 (draft)**

http://www.estelle.org

Estelle is an ISO Formal Description Technique, i.e. a specification language, for concurrent distributed systems. Compare [LOTOS], a companion ISO standard, and [SDL], an ITU standardized language, with which it has some commonality. Estelle is based on modules and interaction points, and uses the asynchronous (non-blocking) send for intermodule interaction, and also shared variables.

Estelle is championed by the LOR, département LOgiciels-Réseaux, (Department of Network Software)

http://www-lor.int-evry.fr/

LOR has produced EDT = Estelle Development Toolset.

For a tutorial, see [Budkowski].

[FDR] **Failures Divergences Refinement checker**

A CSP-based model checker from *Formal Systems Europe:*

http://www.fsel.com/

A companion tool is [Probe].

[GOTCHA-TCBeans] **http://www.haifa.il.ibm.com/projects/verification/gtcb**

A proprietary IBM tool "designed to assist testers in developing, executing and organizing function tests direct against Application Program Interfaces (APIs) and software protocols written in Java, C or C++". The tool has been used in the [Agedis] project. The test process is one of producing a state machine model of system specifications from which an abstract test suite is generated by GOTCHA. This is

translated into test scripts by TCBEANS which are run via an executor or on-the-fly. (Compare [TorX]). From the file system example, it appears that the user must write switch statements in an imperative language to produce the state machine model, but a UML modelling language has been defined in the [Agedis] project. Non-determinism support is claimed (no details given).

[LOTOS] ISO/IEC standard 8807

LOTOS (Language of Temporal Ordering Specification) is an ISO Formal Description Technique, i.e. a specification language, for concurrent distributed systems. Compare [Estelle]. It has historical connections with CCS and CSP. It is algebraic, using processes, events, ordering operators etc. Synchronisation is by shared events as in CSP. Nondeterminism is implicit in parallelism (various interleavings), or can be specified by offering the same event name more than once with the choice operator `[]` (example from Kenneth Turner, Univ. of Stirling):

```
(eat_out; CHINESE MEAL)[](eat_out; INDIAN MEAL)
```

Many implementations of LOTOS exist. LOTOS has been used as the *explorer* element of the [TorX] tool chain.

[OBJECT GEODE] http://www.telelogic.com

http://www.telelogic.com/products/objectgeode/articles.cfm#simulation

The above downloadable paper describes state-base testing from the perspective of exploring the state space of a model written in SDL (Specification and Description Language): *Automated Test Generation with ObjectGeode Test Composer, Alain Kerbrat.*

Abstract: This paper presents the advanced features provided by ObjectGeode Test Composer, a Test Suite generator for conformance testing of distributed systems:

- Test purposes generation based on structural coverage,
- Test cases generation based on state space exploration,
- Interactive and batch generation,
- Test suite structuring and production

[Petri Nets] A modelling tool with affinities to state modelling, originally submitted by C.A. Petri as *Kommunikation mit Automaten*, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM, Nr 3, 1962. See [Murata] for a thorough review of Petri nets.

[PLTL] Propositional Linear Temporal Logic

A temporal logic originating with A. Pnueli (*The temporal semantics of concurrent programs*, Theoretical Computer Science, 13(1):45-60, 1981), described in [Bérard, p.35].

 © Graham G. Thomason 2003-2004

[Probe] **Process Behaviour Explorer**

A tool to interpret and animate CSP specifications from *Formal Systems Europe:*

http://www.fsel.com/

A companion tool is [FDR].

[PROMELA] **(PROcess MEta LAnguage)**

http://cm.bell-labs.com/cm/cs/what/spin/Man.Quick.html

The language allows for the dynamic creation of concurrent processes. Communication via message channels can be specified to be synchronous or asynchronous. Support is provided by [SPIN], which can perform random or interactive simulations of the system's execution or exhaustive verification of the system's state space (e.g. checking for the absence of deadlocks). PROMELA has been used as the *explorer* in the [TorX] tool chain.

[PTK] **see [BakerP]**

A Motorola in-house tool used to generate conformance tests (SDL or TTCN) from Message Sequence Charts (MSCs) and Process Data Unit specifications (PDUs).

[RATIONAL] **http://www.rational.com/**

http://www.rational.com/products/rose/real_time/rtrose.jsp

From the website on Rational Rose RealTime:

> "Developers of embedded, real-time and network systems software applications develop some of the coolest code for the most technologically challenging products and systems. Because of this, they face several challenges that other development environments don't. Many times, this type of software is highly event-driven, concurrent, and often distributed. Stringent requirements must be met for latency, throughput, and dependability. Capturing and effectively communicating designs for such systems can be tough without the right tools. Rational Rose RealTime for Windows or UNIX is the best solution for accelerating your devices & embedded systems software development projects quickly, easily and completely."

The transition semantics appear to be as described in UML books.

[Reactis] **http://reactive-systems.com**

A graphical tool that supports "a large subset of the discrete-time subset of Simulink and Stateflow". It may also interact with MATLAB for calculations. For Simulink, Stateflow and MATLAB, see http://www.mathworks.com. Simulink is strong in numerical algorithms and is aimed at control systems design, signal processing, and communication systems. Stateflow is the state-transition tool. Apart from many features apparently equivalent to UML statecharts, it supports temporal logic and "schedules transitions and events using temporal operators ("before", "after", "at", "every")." In Reactis, state-transition diagrams are shown graphically, and input events can be selected from a source, the default being random events, which it is

admitted (on the *Getting Started* web pages) may lead to poor coverage. State-transition coverage is indicated by green and red colouring of the diagram. Features appear to be geared to interactive simulation: oscilloscope-like windows showing real-time progress of numerical outputs, variable watching, breakpoints, and stepping through model execution.

[RHAPSODY] **http://www.ilogix.com/**

RHAPSODY is a CASE-tool from I-Logix. From the web-page:

> Rhapsody is an enterprise-wide visual programming environment that allows corporations to build and deploy real-time embedded systems and software applications. Rhapsody is designed and optimized for the special needs of the embedded market. Real-time behavioral semantics, target real-time operating system support, model/code associativity, design-level debugging, and production quality code generation increase developer productivity. Rhapsody customers regularly report design cycle reduction of more than 30%, even on the first project.

The semantics of RHAPSODY (and STATEMATE) are described in [Harel-96].

[RSML] **Requirements State Machine Language**

RSML is Mealy-machine based (actions on transitions). See [Heimdahl] for a description of its semantics, and [Leveson] for its origins. [Von der Beeck] gives the following earlier reference with the same title as [Leveson]:

> N. Leveson, M. Heimdahl, H. Hildreth, J. Reese
>
> Requirements Specification for Process Control Systems
>
> Technical Report 92-106, University of California, USA, 1992.

RSML allows for state arrays. Messages can be sent between separate state machines. It supports timing functions. The semantics allow for looping round transitions. Although developed as a specification *language*, a simulator for RSML has been built by Heimdahl.

[SDL] **Specification and Description Language**

This language has been standardized by the ITU (International Telecommunications Union) as ITU-Z.100 and Z.105. It uses ***asynchronous*** message (=signal) passing between processes. It supports objects and inheritance. The basic graphical symbols represent the following items: *state*, message *output* (send), message *input* (consume), message *save* (if not consumed), *task* (perform some action). The notation is convenient for constructing a state transition diagram in small, page-sized portions at a time. Nondeterminism can arise where different interleavings of message arrival are possible.

[SMV] **Symbolic Model Verifier**

A model checking tool developed by K.L. McMillan under the guidance of E.M. Clarke at Carnegie-Mellon University. It uses CTL* as its temporal logic language

 © Graham G. Thomason 2003-2004

(see [Emerson], and uses *binary decision diagrams* in its implementation. Summarised in [Bérard, Ch.12].

SMV is available from

http://www.cs.cmu.edu/~modelcheck/smv.html

The following site is a tutorial and gives an example of modelling a semaphore:

Model checking lecture notes by Marsha Chechik (U. Toronto)

www.cs.toronto.edu/~chechik

[SPIN] **A simulation and verification tool.**

SPIN was mainly developed by G.J. Holzmann at Bell Labs. The following site gives a general description, many theoretical references, workshop information etc.

http://netlib-bell-labs.com/netlib/spin/whatisspin.html

From **[Bérard, p.139]**: SPIN was designed for simulation *and* verification of distributed algorithms. The systems must first be described in [PROMELA]. Spin has two modes: (1) simulation (2) property-checking (using PLTL). Key feature: state space reduction mechanisms, on-the-fly verification and hashing (allowing it to work with 10^7+ states). SPIN was used in the [TorX] tool chain for on-the-fly conformance testing in the Côte de Resyste project (also ref. [Torx]), using a PROMELA description of the model, supporting nondeterminism.

[Stateflow] **see [Reactis]**

[STATEMATE] **http://www.ilogix.com/**

STATEMATE is a statechart system from I-Logix. From the web-page:

> I-Logix' Statemate MAGNUM is the most comprehensive graphical modeling and simulation tool for the rapid development of complex embedded systems. Statemate MAGNUM provides a direct and formal link between user requirements and software implementation by allowing the user to create a complete, executable specification. Operating on an engineering workstation or PC, Statemate MAGNUM creates a visual, graphical specification that clearly and precisely represents the intended functions and behavior of the system being specified. This specification may be executed, or graphically simulated, so the system engineer can explore what if scenarios to determine if the behavior and the interactions between system elements are correct. These scenarios can be captured and included in Test Plans which are later run on the embedded system to ensure that what gets built meets what was specified. This executable specification is also used to communicate with the customer or end user to confirm that the specification meets their requirements.

The semantics of STATEMATE are described in [Harel-96].

Harel's statecharts and I-Logix's STATEMATE differ from UML's interpretations. Even Rhapsody, from I-Logix, conforms to the UML view. The main differences are

1) The form of parallelism allows for variables to be altered in one place, but retain their original value when used in another place. UML assumes a specific sequence.

2) Harel (and CHSM) prioritize giving the outermost transitions on the same event priority; UML takes an object-oriented derived-class-overrides view and gives the inner transition priority:

[TGV] **http://www.irisa.fr/pampa/VALIDATION/TGV/TGV.html**

"TGV (Test Generation with Verification technology) is a prototype for the generation of conformance test suites for protocols. It is based on the model of input/output (labelled) transition systems (IOLTS) and uses algorithms coming from verification technology. TGV has been developed in collaboration with Vérimag Grenoble and uses libraries of the César-Aldébaran Distribution Package (CADP) developed by Verimag Grenoble and VASY from Inria Rhône Alpes. A first prototype has been connected to the GEODE tool (Verilog) and allows the production of test suites in the TTCN format (Tree and Tabular Combined Notation) from SDL specifications." [Du Bousquet] describes the use of TGV in conjunction with [TorX], for random testing.

[TorX] **Côte de Resyste project: http://fmt.cs.utwente.nl/CdR**

TorX tool: http://fmt.cs.utwente.nl/tools/torx/torx-intro.1.html

TorX comes from the Côte de Resyste (COnformance TEsting of REactive SYSTEms) project, a research and development project (1998-2002) funded by the Dutch Technology Foundation STW (http://www.stw.nl/). It is a collaboration between:

- the University of Eindhoven (http://www.tue.nl)
- the University of Twente (http://www.utwente.nl/)
- Philips (http://www.philips.com)

It aims to develop methods and techniques to build a tool for specification-based testing in an automated way based on formal methods. Based on formal testing theory and languages (LOTOS, SDL, TTCN, PROMELA...), the approach is the *Labelled Transition System* one, with a partition between outputs and (always enabled) inputs.

It defines conformance of an implementation i to a specification s as:

- $i \textbf{ ioco } s =_{def} \forall \sigma \in \text{Straces}(s) : out(i \textbf{ after } \sigma) \subseteq out(s \textbf{ after } \sigma)$

Tretmans explains this as: i **ioco**-conforms to s iff

- if i produces output x after trace σ , then s can produce x after σ
- if i cannot produce any output after trace σ, then s cannot produce any output after σ, (quiescence).

A test suite T is *sound* if $i \textbf{ ioco } s \Rightarrow i \textbf{ passes } T$.

A test suite T is *exhaustive* if $i \textbf{ passes } T \Rightarrow i \textbf{ ioco } s$.

TorX is a tool chain, supporting *on-the-fly* testing, consisting of an Explorer-Primer-Driver-Adapter-IUT, as follows:

 © Graham G. Thomason 2003-2004

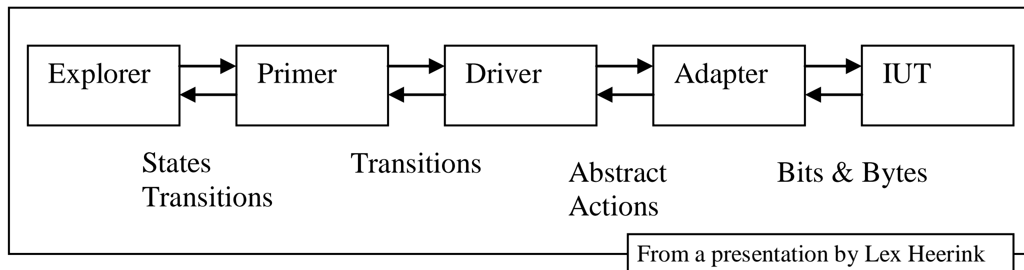


Figure 1. The TorX tool chain

[TVEC] **www.t-vec.com**

A commercial set of tools integrating requirements and test, listed by the [Agedis] consortium. One mode of testing is model-based testing. The web pages do not elaborate on models supported, (UML dynamic models?). The T-VEC "tabular modeler" is derived from the US Naval Research Center's SCR (Software Cost Reduction) model, which is a requirements formalism, amenable to model checking, e.g. by SPIN.

[UML] **http://www.omg.org**

(The Object Management Group Website)

UML specifications (v. 1.5, November, 2003) are available from the website.

Section 2.12 is on State Machines, which are a subpackage of the Behavioral Elements Package, which also includes Collaborations, Use Cases and Activity Graphs.

UML is a visual *modelling* language rather than a visual *programming* language [section 1.5.1.1, pp.1-7], so a direct comparison with STATECRUNCHER is not always possible. STATECRUNCHER is close to UML in semantics, and it is certainly our aim to align STATECRUNCHER as precisely as possible with UML if we have the opportunity for future developments. We note the following features of UML:

- *Change events* (lambda transitions), e.g. transitions triggered by data taking on a certain value. There are semantic issues as to when data is allowed to trigger such a transition.
- *Deep history and shallow history vertices* (i.e. as transition targets, also known as pseudo-states, so that different transitions can target a composite state individually invoking deep history, shallow history or no history). These are on STATECRUNCHER's wish-list.
- *Joins, forks, junctions and choices*. STATECRUNCHER can accommodate *joins* using the `in(...)` function as a guard. STATECRUNCHER has *forks* (the split operator). STATECRUNCHER can implement the functionality of *junctions* and *choices* using multiple transitions.
- *Deferrable events*. Not supported in STATECRUNCHER.
- *Do Activities*, describing processing associated with being in a state.

- *Synch states*, used for ordering forks and joins.
- *Time Events*. Such an event can express expiry of a deadline. STATECRUNCHER does not have any special constructs for expressing time.
- *Firing priorities*. Transitions originating from a substate has priority over a transition originating from any of its containing states. STATECRUNCHER now (Release 1.03 and higher) conforms to this.

[VVT-RT] **Validation, Verification and Test of Real Time Systems**

A tool from Verified Systems International GmbH, Bremen, in co-operation with the Bremen Institute of Safe Systems (BISS) within the Center for Computing Technology (TZI) at Bremen University. It is based on CSP [Hoare]. For a paper on an application of it, see [Schlinghoff].

 © Graham G. Thomason 2003-2004

4. Publications relating to verification, testing and/or state machines

[Alhir] **Sinan Si Alhir**

UML in a Nutshell

O'Reilly & Associates., 1998. ISBN 1-56592-448-7

This book contains intense, concise detail on UML. Chapter 11 covers statechart diagrams. It elaborates on compound transitions (decision branching), and splitting/synchronizing control.

[BakerP] **Paul Baker, Paul Bristow, Clive Jervis, David King and Bill Mitchell**

Automatic Generation of Conformance Tests from Message Sequence Charts

System and Software Engineering Research Lab (UK), Motorola Labs

The paper describes how the PTK tool (Motorola proprietary) is used to generate conformance tests from Message Sequence Charts (MSCs) and Protocol Data Unit specifications (PDUs). PTK generates SDL of TTCN scripts. Interleaving semantics of MSCs are used to generate all traces of events. Nondeterminism is handled by generating separate scripts for separate outcomes, with one precise outcome giving a test result of *pass*, and alternatives giving a test result of *inconclusive*. This makes it possible to check that all nondeterministic outcomes have been obtained (but it is not explained how they might be stimulated).

[BCS-SIGIST] **Standard for Software Component Testing**

British Computer Society - Special Interest Group in Software Testing

This document contains a great number of definitions and descriptions of testing terms and metrics. It defines State Transition Coverage as follows: For single transitions, the coverage metric is the percentage of all valid transitions exercised during the test. This is known as 0-switch coverage. For *n* transitions, the coverage measure is the percentage of all valid sequences of n transitions exercised during the test. This is known as (n-1) switch coverage.

[Beizer] **B. Beizer**

Software Testing Techniques, 2nd edition

International Thomson Computer Press, 1990, ISBN 1850328803

A very good introduction to practical software testing in general, covering various testing techniques. Chapter 11 is on States, State Graphs, and Transition Testing. The book introduces a tabular representation of transitions. It contains good advice on what to model (p.389). All examples are presented as flat deterministic finite state machines.

[Belinfante] **Axel Belinfante**

Formal Test Automation: A Simple Experiment

(A [TorX] / Côte de Resyste report)

This paper describes TorX in use, with test scenarios specified in LOTOS, PROMELA and SDL, testing a conference protocol.

[Bérard] **B. Bérard**

Systems and Software Verification

Springer-Verlag, 2001. ISBN 3-540-41523-8

This excellent book describes in turn automata, temporal logic, model checking, symbolic model checking, and timed automata. It is concerned with *model checking*, i.e. proving properties of a model, (so verifying a *design*), rather than *testing a model against an implementation*. The temporal logic languages CTL* and PLTL are used. Amongst the tools described are: SMV, SPIN and Design/CPN, (and some timed/real-time tools).

[Binder] **Robert V. Binder**

Testing objects: State-based testing: Sneak paths and conditional transitions

Object Magazine, October 1995, pp. 87-89

This article illustrates the practical need to test an object (it also applies to a *system*) with messages that should not be accepted (what STATECRUNCHER calls *non-transitionable events*), and to check that the state has not changed. This is, of course, in addition to normal transitioning tests. A bank account example is given. Code which allows an illegal transition is a called a *sneak path*; it could be deliberate for the purposes of theft or sabotage. An equivalent situation arises with transitions having a condition that evaluates to *false*. There is a discussion on how to handle illegal messages at a coding level.

 © Graham G. Thomason 2003-2004

[Bogdanov] **Kirill Bogdanov and Mike Holcombe (Univ. of Sheffield)**
Statechart testing method for aircraft control systems
Software Testing, Verification and Reliability, 2001; *11*:39-54

The authors take a statechart model of an aircraft control system with commands *climb, descent, flaps_down, flaps_up, terminate, level*. The approach requires a deterministic specification and implementation. Unlike the STATECRUNCHER case, events can be combined and negated in labelling a transition: *command* ^ ¬ *terminate*. The approach is a black-box one, because states are distinguished using a *characterisation set*, described here as a path which exists from one state but not another.

[Booch] **Grady Booch, James Rumbaugh, Ivar Jacobson**
The Unified Modelling Language User Guide
Addison Wesley, 1999. ISBN 0-201-57168-4

A tutorial by the original developers of UML. Chapters 21 and 24 are on State Machines and Statechart Diagrams.

[Brinksma] **Ed Brinksma**
Testing Transition Systems: An Annotated Bibliography.
University of Twente, The Netherlands, Formal Methods and Tools Group.
http://fmt.cs.utwente.nl

This paper covers developments in formal testing theory and formal test generation. Test generation products mentioned: TVEDA, TGV, TestComposer (SDL-based; all have fed into OBJECTGEODE); VVT-RT (which uses CSP), SaMsTaG and AUTOLINK (which derive tests from SDL).

[Bruns] **Glenn Bruns**
Distributed Systems Analysis with CCS
Prentice Hall 1997, ISBN 0-13-398389-7

A book that teaches CCS with many examples (arbiters, triple-modular redundancy and others). Complementary to [Milner], which is the authoritative text.

[Budkowski] **A. Budkowski, P. Dembinski, M. Diaz**
ISO Standardized Description Technique Estelle

This is a tutorial on [Estelle], available from
http://www-lor.int-evry.fr/idemcop/uk/est-lang/download/short-estelle-tutorial.pdf

[Büssow] **Robert Büssow, Robert Geisler, Wolfgang Grieskamp, Marcus Klar**
The μSZ Notation Version 1.0

The **μSZ** notation is used in the [ESPRESS] project. It combines Z and Harel-style statecharts. Process classes are: *data space* (variables), *operational behaviour* (statechart structure and transitions), *behavioural constraints* (can be specified with a temporal logic), *structural embedding* (aggregations of instances of classes).

[Chow 78] **Tsun S. Chow**
Testing Software Design Modeled by Finite-State Machines
IEEE Transactions on Software Engineering, Vol SE-4, No 3, May, 1978

An early paper on obtaining and measuring state coverage. Discusses the use of ***P***, a set of input sequences to take a machine to every source state of a transition and to trigger that transition. ***P*** can be obtained from ***T***, a testing tree, which is a recursive exploration of the state space from everywhere not seen before. Discusses further ***W***, the characterization set, a set of input sequences capable of distinguishing the behaviours of every pair of states in a minimal finite state machine.

[Component+] **Built-in testing for Component-based Development**
EC IST 5th Framework Project IST-1999-20162 Component+
http://www.component-plus.org

This project aims at making software component systems self-testable and run-time using Built-In Testing (BIT) facilities. These facilities are structured as additional interfaces to the components, a *provides* interface to test and a *requires* interface to notify. A tester component might contain corresponding interfaces that are bound to both of these interfaces. A small extra size overhead in the components is regarded as acceptable, as in the case of VLSI chips. Both interface contract and quality of service (QoS) can be tested. QoS testing is continuous verification against e.g. deadlock, time constraint violation, data corruption, user conformance, memory leaks or conflicts. An example of contract testing is actually *state transition testing*, in this case, of a stack (sections 3.3.1.1 - 3.3.1.2 of the *Deliverable D3* document).

[Dahbura] **Anton T. Dahbura, Krishnan K. Sabhani, and M. Ümit Uyar**
Formal Methods for Generating Protocol Conformance Test Sequences
Proceedings of the IEEE, Vol. 78, No. 8, August, 1990

The context is FSMs. This paper gives an overview of the four main methods of generating test sequences for such deterministic FSMs: (1) the *transition tour* (the T method), (2) *distinguishing sequences* (the D method), (3) *characterizing sequences* (the W-method) and (4) *unique I/O sequences* (the U method). These are illustrated by worked examples.

 © Graham G. Thomason 2003-2004

[de Vries] **René de Vries and Jan Tretmans**
On-the-fly Conformance Testing Using SPIN
Formal Methods & Tools Group, University of Twente, The Netherlands

A Côte de Resyste report (see[TorX], [Tretmans]), and so *labelled transition system* based. [SPIN] is used with PROMELA specifications, allowing for large state spaces. Nondeterminism is handled in an *on-the-fly* algorithm (section 3). Quiescence (no output) is also accepted if it is valid.

[Du Bousquet] **Lydie Du Bousquet, Solofo Ramangalahy, Séverine Simon, César Viho**
Formal Test Automation: The Conference Protocol with TGV/TorX
Available on the web at the [TorX] site.

This paper describes the first experiment with [TGV] and [TorX] in combination. The system tested was a multicast protocol implementation (a kind of chatbox), specified in LOTOS. Manually generated and random testing were compared. An on-the-fly technique was used. Of 25 mutant systems (i.e. with seeded errors), manual testing found all but one. Random testing found all mutants.

[Dupuy] **Arnaud Dupuy and Nancy Leveson**
An Empirical Evaluation of the MC/DC Coverage Criterion on the Hete-2 Satellite Software
DASC (Digital Aviation Systems Conference), October 2000

This paper argues for the testing effectiveness of obtaining the boolean expression coverage criterion known as MC/DC (Modified Condition / Decision Coverage), as defined in the USA Department of Defense standard DO178B. In this standard, test cases are generated such that each term in the expression is shown to be capable of independently affecting the value of the whole expression. For an application in state-based testing, see [Offutt].

[Eilenberg] **Samuel Eilenberg**
Automata, Languages, and Machines
Academic Press, New York, 1974

Chapter X *Machines* is the seminal publication on X-Machines. These are state machines that operate on data of type X as they transition.

[Emerson] **E.A. Emerson and J.Y. Halpern**
“Sometimes” and “Not Never” revisited:
On branching versus linear time temporal logic.
Journal of the ACM, Vol. 33, Nr. 1, pp. 151-178, 1986.

Describes the CTL* language, representing a model checking logic, (used in the [SMV] tool). The underlying concepts of linear time and branching time had already been described in a paper by L. Lamport, cited (“Sometime” is sometimes “not never”,—On the temporal logic of programs, in *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, Las Vegas, Nev., Jan 28-30. ACM. New York, 1980, pp.174-185). Lamport's concepts are extended and critiqued, resulting in a unified approach, embodied in the language CTL*.

[Farchi] **E. Farchi, A. Hartman and S.S. Pinter**
Using a model-based test generator to test for standard conformance
IBM Systems Journal, Vol. 41, Nr. 1, 2002.

This article describes state-based testing of a stack, a file system and a Java exception handler, and how the state explosion problem was avoided by using projection state and projection transition coverage as a means of specifying test criteria.

[Fuhrmann] **Kay Fuhrmann, Jan Hiemer**
Formal Verification of STATEMATE Statecharts

An [ESPRESS] publication. A technique is given whereby STATEMATE statecharts are translated into CSP for verification with the [FDR] model checking tool. The hard part appears to be the translation of STATEMATE's step semantics.

[Fujiwara 91] **Susumu Fujiwara, Gregor v. Bochmann**
Test Selection Based on Finite State Models
IEEE Transactions on Software Engineering, Vol. 17, No 6, June 1991

The context is principally deterministic FSMs. The paper presents an optimization to the *W method* (see [Chow]), called the *partial W method*. The optimization is based on using an *identification set* to identify a state, rather than the *characterization set*. The identification set is a state-dependent subset of the characterization set. (If an identification set consists of a *single* sequence, it is equivalent to a UIO approach). Good worked examples are given. There is a discussion of the following testing issues: (A) implementations having more states than the specification, (B) issues arising from incomplete specifications, (C) synchronization of distributed systems, (D) specifications including data flow, (E) nondeterministic implementations and/or specifications, and (F) OSI protocol conformance testing.

 © Graham G. Thomason 2003-2004

[Fujiwara 92] **Susumu Fujiwara, Gregor v. Bochmann**
Testing non-deterministic state machines with fault coverage
Protocol Test Systems IV, J Kroon et al. (editors)
Elsevier Science Publishers B.V. (North-Holland), 1992

The paper presents a test selection method for testing nondeterministic systems. The approach is the *labelled transition system* one, not the *finite state machine* one. A successful test run proceeds through all actions specified without deadlocking.

[Gurevich] **Yuri Gurevich**
Sequential Abstract State Machines Capture Sequential Algorithms
Microsoft Research report MSR-TR-99-65
Also published as: **ACM Transactions on Computational Logic,**
vol. 1, no. 1, July 2000, 77-111

Sequential algorithms are related to Abstract State Machines by a correspondence between variable values and abstract state, though these states can be interpreted as structures of mathematical logic, and as memory. States are transformed in computation steps, which are related to transitions. Nondeterminism is seen as the environment making a choice. “Nondeterministic algorithms are special interactive programs (section 9).” [ASML] is a tool embodying the notions of Abstract State Machines. As with ASML, the nature of Abstract State Machines as described has an *imperative* rather than *reactive* character, (reinforced by the examples of Eratosthenes' sieve and Euclid's greatest common divisor algorithms).

[Harel 87] **D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman**
On the Formal Semantics of Statecharts
Logic in Computer Science, 2nd Annual Conference, 1987, pp.54-64

This paper has effectively laid the foundations for modern approaches to state modelling. It elaborates on the concept of ‘statecharts’ (as opposed to flat state diagrams) which Harel had recently introduced [D.Harel, *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming, 8, 1987]. Harel's statecharts have XOR (called OR in [Harel 96]) and AND components, default states, history, and broadcast events. The paper discusses the semantics of statecharts using the concept of *micro-steps*, discussing such difficulties as the value of shared variables that can, in principle, be assigned simultaneously possibly conflicting values. Nondeterministic situations are recognized, and some constructs are introduced to resolve them to a deterministic course of action. The concepts of this paper led to the commercial product STATEMATE.

The paper underlies [CHSM] and so indirectly also STATECRUNCHER.

[Harel 96] **David Harel and Amnon Naamad**

The STATEMATE Semantics of Statecharts

ACM Transactions on Software Engineering and Methodology 5:4, October 1996

This paper gives the semantics that the I-Logix products [STATEMATE] MAGNUM and [RHAPSODY] employ. The products can be used for testing and for code synthesis. A notable feature is prioritized transitions. Nondeterminism is handled as follows.

- Conflicting transitions [pp.16-17] (STATECRUNCHER's fork nondeterminism) result in the generation of sets of steps (transitions and static reactions, the latter being equivalent to additional transitions). The selection can be carried out interactively by the user, or by specifying a selection criterion at the start. The dynamic tests tool will try out all the different possibilities in an exhaustive fashion. The code synthesized by the software code generator will select the first possibility it finds that is enabled and will proceed to execute it.
- Racing conditions [pp. 24-25]. Where there are multiple orderings (such as Fig. 25, where t2 and t3 race), the paper states that STATEMATE reports a racing condition.

[Heimdahl 96] **Nats P.E. Heimdahl and Nancy G. Leveson**

Completeness and Consistency in Hierarchical State-Based Requirements

IEEE Transactions on Software Engineering, Vol 22, No 6, June 1996

The paper addresses *completeness* and *consistency* in a statechart. Statecharts are modelled as functions. The language used is [RSML], which is Mealy-machine based (actions on transitions). Robustness is defined by [p.363]: (1) every state must have a behaviour (transition) defined for every input; (2) the logical OR of the conditions on every transition out of any state must form a tautology; (3) every state must have a timeout. This is called ***d-completeness***. The transition *relation* is made to behave as a *function*. In this way determinism is imposed in d-completeness. Completeness checking is maintained in composition of state machines.

[Hennie] **F.C. Hennie**

Fault Detecting for Sequential Circuits

Proceedings of the 5th Annual Symposium on Switching Theory and Logical Design, 1964, pp. 95-110.

The approach is Mealy FSMs, though in the guise of circuits that take inputs of 0 or 1 and produce outputs of 0 or 1. It is an early paper introducing and synthesizing distinguishing sequences as a means of state checking.

[Hierons 98] **Rob M. Hierons**

Adaptive testing of a deterministic implementation against a nondeterministic finite state machine

The Computer Journal, *41*, 5 pp 349-355

Available from the author's home page: www.brunel.ac.uk/~csstrmh

This paper shows how an implementation that is known to be a deterministic state machine can be tested against a nondeterministic model of it. The paper introduces *d-distinguishing* sequences, that distinguish two states in an NFSM provided the implementation is deterministic (although it is not known how). On-the-fly tests learn from the observed behaviour and so adapt the test generation accordingly.

STATECRUNCHER, in conjunction with other programs communicating with it, could be of assistance in implementations of algorithms like this, perhaps by exploring a nondeterministic UML model and helping find *d-distinguishing* sequences.

[Hierons 03] **Rob M. Hierons**

Generating Candidates when testing a deterministic implementation against a Non-deterministic Finite State Machine

The Computer Journal, 46, 3, pp. 307-318

The paper addresses the problem of testing an implementation that is known to be deterministic against a nondeterministic specification. A *candidate* is a deterministic FSM that is generated from the nondeterministic specification and the implementation. It has the property that if the implementation conforms to the candidate, the implementation conforms to the specification. Tests can then be derived from the candidate, using test generation algorithms for deterministic FSMs.

[Hoare] **C.A.R. Hoare**

Communicating Sequential Processes,

Prentice Hall International Series in Computer Science, 1985.

ISBN 0-13-153271-5 (0-13-153289-8 Paperback)

This book describes CSP, (Communicating Sequential Processes): a process algebra (or calculus) for specifying state behaviour in terms of processes and events. There are various operators for parallel composition of processes. Ordinary engagement of two or more processes is based on sharing of events in their ‘alphabet’. There are operators (n, o) for nondeterministic compositions. Algebraic laws enable rewriting, simplification and comparison of process expressions.

[Hong 95] **Hyoung Seok Hong, Jeong Hyun Kim, Sung Deok Cha and Yong Rae Kwon (Dept. Computer Science, Korea Advanced Inst. of Science & Technology)**
Static Semantics and Priority Schemes for Statecharts
Proceedings of COMPSAC '95, IEEE Computer Society Press.

This paper defines static semantics of statecharts and identifies types of nondeterminism. The semantics allow for conjunctions two or more simultaneous events and their negations, e.g. $\alpha\wedge\neg\beta$, (unlike STATECRUNCHER). Nondeterminism in a statechart is identified as:

- *external* nondeterminism, where with two simultaneous events the system can have differing resultant states.
- *internal* nondeterminism, where there are different resulting states after processing one event.

The paper also discusses invalid transitions, with formal properties for valid transitions, and the use of priority when there are simultaneous events.

[Jagadeesan] **L.J. Jagadeesan, A. Porter, C. Puchol, J.C. Ramming, L.G. Votta**
Specification-Based Testing of Reactive Software:
Tools and Experiments. Experience report,
Proc. of the International Conference on Software Engineering,
May 1997

This paper describes an unusual combination of *model checking* and *implementation testing*. A temporal logic specification is made of the system, defining safety properties. From this, finite state machines (FSMs) that accept input-output traces that violate the safety properties are automatically generated. From the FSMs, test inputs are generated, and the IUT is checked for whether the safety properties are violated by these inputs, and if so, an alert is given. The specification may be nondeterministic, but this is not elaborated on. Examples given: an elevator system and a telephone switching system.

[Kloosterman] **Hans Kloosterman**
Test derivation from non-deterministic finite state machines
Protocol Test Systems, V (C-11), G. v. Bochman et al. (editors),
Elsevier Science Publishers B.V. (North-Holland), 1993.

This paper describes “algorithms for the generation of test sequences from non-deterministic finite state machines (NFSMs). The test sequences are synchronizing sequences (SS), transferring sequences (TS) and unique input/output (UIO) sequences.” An SS may not exist, but in practice for protocols they usually do. Compared to a (strongly connected) deterministic situation, the following issues arise: A TS does not always exist because it may not always be possible to transfer deterministically to this state. The UIO has to check a set of states, not just one

© Graham G. Thomason 2003-2004

expected state. The SS and TS can be regarded executing the test and the UIO as getting extra output to verify the result.

[Kwan] **Kwan Mei-Ko**
Graphic programming Using Odd or Even Points
Chinese Mathematics 1962, Vol. 1, pp. 273-277.

The paper shows how to generate a postman's route, i.e. a transition tour. The *Chinese postman* problem is so named in honour of the author.

[Lee 96] **David Lee and Mihalis Yannakakis**
Principles and Methods of Testing Finite State Machines
Proceedings of the IEEE, Vol. 84, No 8, August, 1996

This paper gives a good overview of testing based on Mealy machines (actions on transitions, not on state exit/entry). The paper states explicitly that it does not cover validation and verification (model checking), which are distinct from testing. Key concepts: *distinguishing sequence* of events to identify states; *unique input/output (UIO) sequence* of events to verify some particular state; *checking sequence* to test for conformance of a black box to it specification. The paper also describes *characterization sets* (see [Chow]) which distinguish pairs of states, and *transition tours* (see the Philips report [Raptis 98]). Nondeterminism is mentioned, but the main exposition focuses on deterministic machines.

[Leveson] **N.G. Leveson, M.P.E. Heimdahl, H. Hildreth, J.D. Reese**
Requirements Specification for Process Control Systems
IEEE Transactions on Software Engineering, vol. 20, no. 9, Sept 1994

The paper describes how the need for a specification language for safety-critical systems led to [RSML], and describes RMSL semantics. RSML is based on Harel's statechart notation, with some omissions where the complexity did not warrant them, and some extensions to allow for the requirements needing to be expressed. The application considered is an aircraft collision avoidance system. A simulator for RSML was built by Heimdahl.

[Li] **J Jenny Li, Hong Liu, Rudolph E. Seviora**
Constructing Automated Protocol Testing Oracles to Accommodate Specification Nondeterminism
Sixth International Conference on Computer Communications and Networks (ICCCN '97), September 22 - 25, 1997, Las Vegas, NV

The paper describes an SDL-based implementation of a nondeterministic test oracle. For *local* nondeterminism (like a STATECRUNCHER a fork), a construct *ALL* that supports *AND-states* is introduced, a counterpart to *ANY* in the specification. (AND-states are alternative nondeterministic states, not Harel's parallel states of the same

designation). For global nondeterminism (like a STATECRUNCHER race), permutations of signal arrival orders are needed, apparently also handled by the AND-states. The method was trialled with a small protocol serving 60 nodes. Test generation was random testing. The maximum number of 'concurrent' states generated was 1442.

[Lüttgen 00] **Gerald Lüttgen, Michael. von der Beeck and Rance Cleaveland**
A Compositional Approach to Statecharts Semantics
Presented at FSE (Foundations of Software Engineering) 2000, San Diego
http://www.cs.virginia.edu/fse8/
Available from Cleaveland
http://www.cs.sunysb.edu/~rance/publications/./2000.html

The paper discusses the semantics of a statecharts composed of smaller statecharts. The approach is one of micro-step semantics as in [Harel], on the ticking of a global clock, from which the macro-composition is recovered, (rather than the sequenced approach of UML). It also has the concept of more than one conjoined event, or absence of an event, (e.g. $a \wedge \neg b$), on a transition.

[Milner] **Robin Milner**
Communication and Concurrency
Prentice Hall, 1989. ISBN 0-13-114948-9

This book describes CCS: the Calculus of Communicating Systems, a process algebra (or calculus), for specifying state behaviour in terms of processes and events. Ordinary engagement of two processes (no more than that) is based an event and its complement being possible, giving rise to a *possible* internal transition τ, (so introducing potential nondeterminism). The ordinary summation operator, (+), specifies alternative behaviours, which may include nondeterministic choices on the same event. Algebraic laws enable rewriting, simplification and comparison of process expressions.

[Murata] **Tadao Murata**
Petri Nets: Properties, Analysis and Applications
Proceedings of the IEEE, Vol 77, No 4, April, 1989

The paper gives a comprehensive survey of what the title proposes, with 315 references. Petri nets can be used to model deterministic and nondeterministic finite state machines [p.544]. Property checking (of Petri nets themselves rather than state machines) is discussed (e.g. liveness and safety) [p.550, p.555]. Many applications apart from state-machine related ones are discussed. Higher level nets, including coloured Petri nets (for which an implementation now exists, see [Design/CPN]), are described.

 © Graham G. Thomason 2003-2004

[Myers 79] **G.J. Myers**
The Art of Software Testing
John Wiley & Sons, 1979. ISBN 0-471-04328-1

This is an early, but still popular, book on standard software testing techniques. It is strong on *cause-effect graphing* (in Chapter 4), a major complementary testing technique to state-based testing. A future research area will probably be to combine cause effect graphing and state based modelling, perhaps in connection with parameterized events.

[Offutt] **Jeff Offutt**
Generating tests from UML specifications
George Mason University, Fairfax VA 22030, USA
http://isse.gmu.edu/faculty/ofut/rsrch/papers/uml99.pdf

This paper describes a tool called UMLTEST, which takes Rational Rose UML specifications of state machines, requiring that they be deterministic, and generates test cases at *full predicate* and transition *pair coverage* level. By *full predicate*, the author means that the guard (or enabling) condition on the transition is exercised according to a boolean expression coverage criterion known as MC/DC (Modified Condition / Decision Coverage), as defined in the USA Department of Defense standard DO178B. In this standard, test cases are generated such that each term in the expression is shown to be capable of independently affecting the value of the whole expression. The tool was empirically evaluated against a cruise control system with seeded faults, all of which were found, which was better than with just transition pair or statement coverage testing.

[Ostroff 89] **Jonathan S. Ostroff**
Temporal Logic for Real-Time Systems
John Wiley & Sons Inc, 1989. ISBN 0 08380 086 6

The book describes ESMs (Extended State Machines), which, unlike statecharts, contain communication channels over which events are executed, Manna-Pnueli temporal logic, RTTL (Real Time Temporal Logic), and a proof system associated with this, PS-RTTL. The perspective is property checking, not testing.

[Petrenko] **Alexandre Petrenko, Nina Yevtushenko, Alexandre Lebedev, Anindya Das**
Nondeterministic State Machines in Protocol Conformance Testing
Protocol Test Systems VI (C-19), pp. 363-378, 1994

This paper describes test suite generation for NFSMs, introducing the concept of *r-distinguishing* sequences to distinguish states in an observable NFSM.

[Phadke] **Madhav S. Phadke**
Planning efficient software tests
http://www.stsc.hill.af.mil/crosstalk/1997/10/planning.asp

This is a popular article explaining *orthogonal arrays.* Suppose a routine needs testing with 4 parameters, (A,B,C, and D), each of which can take 3 values (1,2, and 3). Exhaustive testing would require running 3^4=81 tests. But suppose we find it adequate that all pairwise parameter value combinations are taken. A table can be found satisfying this with 9 entries of values of the 4 parameters as follows:

```
ABCD
1111
1223
1332
2122
2231
2313
3133
3212
3321
```

For *pairwise* coverage we speak of orthogonal arrays of *strength 2*. If we had required that all *triples* of parameters should be covered for all combinations of values, the strength would be 3 and so on. See [Sloane] for libraries of orthogonal arrays; the above array is equivalent to the one at http://www.research.att.com/~njas/oadir/oa.9.4.3.2.txt. (There is opportunity to combine orthogonal array techniques with state-based testing where there are parameterized events).

[Robinson 00] **Harry Robinson**
Intelligent Test Automation
Software Testing and Quality Engineering, Sept/Oct 2000, pp. 24-32

This popular article makes the practical case for model-based testing using four amusing cartoons.

[Robinson www] **Harry Robinson**
Model Based Testing Home Page (maintained by)
http://wwwgeocities.com/model_based_testing

This is a popular website with many articles on model-based testing.

[Sabnani] **Krishnan Sabnani and Anton T. Dahbura**
A Protocol Test Generation Procedure
Computer networks and ISDN Systems 15 (1988), pp. 285-297

The context is Mealy FSMs. The paper describes the UIO (unique I/O sequence) method of checking states, so that the target state of all transitions can be checked.

 © Graham G. Thomason 2003-2004

[Schlinghoff] **Dr Holger Schlinghoff, Oliver Meyer, Thomas Hülsing**

Correctness Analysis of an Embedded Controller

http://www.informatik.hu-berlin.de/~hs/Publikationen/

pointing to

http://www.informatik.hu-berlin.de/~hs/Publikationen/1999_DASIA_Schlingloff-Meyer-Huelsing_Correctness-Analysis-of-an-Embedded-Controller.ps

This paper reports on the use of the [VVT-RT] tool to test a safety-critical application: a thermal control unit of the X-ray satellite ABRIXAS. A target system is tested against CSP specifications. All possible execution sequences (presumably of inputs, i.e. events) were executed. The results were to find incomplete parts of specifications and several bugs, including a hardware problem, where EEPROMs did not meet their specification.

[Schneider] **Steve Schneider**

Concurrent and Real-time Systems, The CSP Approach

John Wiley & Sons Ltd, 2000, ISBN 0-471-62373-3

A book on CSP, good for learning CSP, that is complementary to [Hoare], which is the authoritative text.

[Shen] **Y.-N. Shen, F. Lombardi and A.T. Dahbura**

Protocol Conformance Testing Using Multiple UIO Sequences

IEEE Transactions on Communications, Vol. 40, No. 8, August, 1992

In the context of deterministic Mealy FSMs, the paper presents results for test sequences using a transition tour, validating the target state of each transition with a UIO (Unique I/O sequence), built into the tour, with the refinement that the best UIO is chosen (where there are several options), so as to produce an optimised tour.

[Simons] **Anthony J.H. Simons**

On the Compositional Properties of UML Statechart Diagrams

Rigorous Object-Oriented Methods, 2000

"This paper proposes a revised semantic interpretation of UML Statechart Diagrams which ensures, under the specified design rules, that Statecharts may be constructed to have true compositional properties." The example of an automatic gearbox is given, and the issue of concurrent events at different compositional levels is discussed. We remark that in STATECRUNCHER, the issue of concurrent, interrupting or conflicting events does not arise, as any triggered transition is processed to completion as regards state occupancies, before any associated actions, which will have been collected, are processed from a consistent and stable configuration.

© Graham G. Thomason 2003-2004

[Sloane] **N.J.A. Sloane**

A library of orthogonal arrays

http://www.research.att.com/~njas/doc/OA.html

For a description of orthogonal arrays, see [Phadke].

[Stannett] **Mike Stannett and A.J.H. Simons**

Complete Behavioural testing of Object-Oriented Systems using CCS-Augmented X-Machines

Test Report CS-02-04, Dept. of Computer Science, United Kingdom

The paper combines X-Machines and [CCS], generating a new behavioural specification and modelling language, CCS-XM. A form of communicating X-machine, communicating in the CCS sense, not in the shared memory sense, is defined: a Process X-machine (PXM). The analysis of the way PXMs communicate is analogous to STATECRUNCHERs composition mechanism. The paper has:

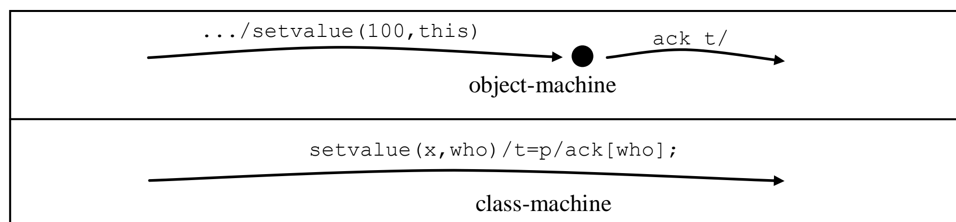


Figure 2. PXM assignmment to a static class variable by an object

The STATECRUNCHER analogue is:

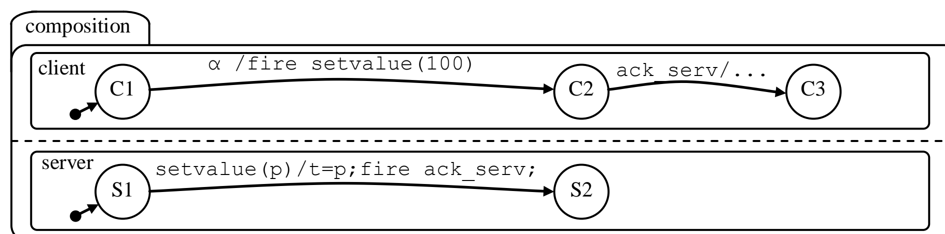


Figure 3. STATECRUNCHER'S composition paradigm making an assignment

Here, we have not made the `ack_serv` event unique to the specific caller as in the paper (the *`this`* keyword). Since this server does not support recursion, the server can only be serving one client at a time, so it is sufficient for `ack_serv` to be unique to the *server*; it cannot then be confused with the acknowledgement from any other server serving a different function. In [StCrFunMod], we propose a composition mechanism for recursive state machines, where the returned acknowledgement need not have a unique name at all, and targets its caller by means of scoping operators.

[Tao Xie] **http://www.cs.washington.edu/homes/taoxie/testingresearchers.htm**

A large list of testing researchers, with web links.

 © Graham G. Thomason 2003-2004

[Tretmans] **Jan Tretmans**

Test Generation with Inputs, Outputs and Repetitive Quiescence

Department of Computer Science, University of Twente

"...A test generation algorithm is given which is proved to produce a sound and exhaustive test suite from a specification, i.e. a test suite that fully characterizes the set of correct implementations". This paper underlies the later [TorX] publications. The approach is the *labelled transition system* one, not the *finite state machine* one. Publications by Jan Tretmans are listed/summarised/downloadable as the case may be at: **http://fmt.cs.utwente.nl/publications/tretmans.pap.html**

[von der Beeck] **Michael von der Beeck**

A Comparison of Statechart Variants

Aachen University of Technology, Aachen, Germany

This paper uses a set of distinctive features to make a detailed comparison of 21 statechart variants. These are: [RSML] (Leveson), [Argos] (Maraninchi), and statecharts indicated by developers/designers only (sometimes with collaborators): Harel, Huizing, Pnueli, Hooman, Classen, Maggioli-Schettini, Day, Peron, Keston, von der Beeck. All but one of these statecharts allows for the *specification* of nondeterminism, but the only description of *handling* of nondeterminism given is to resolve the potential nondeterminism to a deterministic choice.

If we attempt to characterise STATECRUNCHER by von der Beeck's criteria, we have

- ***(1) Perfect synchrony***: Yes, there is no buffering of events, but when one event fires another, output is generated in particular orderings of on-state-exit actions, on-transition actions, on-state-entry actions etc.
- ***(2) Self-triggering***: No. Two transitions triggered by `α/fire β` and `β/fire α` will not spontaneously take place - they require a separate generation of an initial α or β.
- ***(3) Negated trigger event***: No. There is no concept of negated events, or conjunction of events, such as $\alpha \wedge \neg\beta$. Events can only be offered sequentially, and triggered transitions are seen as a set of sequences representing interleaving.
- ***(4) Effect of a transition is contradictory to its cause***: Not applicable, because there is no concept of triggering from a negated event. A transition `¬α/fire α` is not specifiable.
- ***(5) Inter-level transition:*** Yes. Source and (multiple) target states of a transition can all be in at any level in the hierarchy (provided the transition is not illegal).
- ***(6) State reference:*** Yes. This is the `in(...)` function.
- ***(7) Compositional semantics, Self-termination:*** Yes, inasmuch as a client-server paradigm exists for composition, mirroring formal software component composition. The client and server can be tested independently, and the inter-

component events can be hidden by attaching them to a PCO (point of control and observation) that indicates that they are not external events in compositions. Discussed in [StCrSemCom]. ***Self-termination*** is supported, but it is not needed as an inter-level work-around.

- ***(8) Operational versus denotational semantics:*** Denotational, inasmuch as we specify the exact transition algorithm in a computer-independent way, and an abstract-model-independent way.
- ***(9) Instantaneous state:*** Yes. This is the knock-on effect in a chain of transitions, and states are simultaneously entered and exited, regarding the whole chain of execution as being atomic, and so conceptually instantaneous, to the user.
- ***(10) Durability of events:*** No, events are discrete, and have no duration.
- ***(11) Parallel execution of transitions:*** Yes, parallel execution of transitions is supported, but with selectable interleavings. The article regards this feature as being contradictory to (9), but we have explained and qualified our interpretation of these points
- ***(12) Transition refinement:*** Not applicable, because we support instantaneous states, giving the equivalence of transition sequences.
- ***(13) Multiply entered or exited instantaneous state:*** Yes. This is the cycling issue, which we regard as *advantageous* (provided it is bounded), especially in conjunction with nondeterminism, for reasons given in [StCrMain].
- ***(14) Infinite sequence of transition executions at an instant in time***: Not prohibited. A useless infinite loop could theoretically be detected, at the cost of execution time resources (performance and memory). We leave it up to the user not to program an infinite loop, as it were, as is the case in a language such as 'C'.
- ***(15) Determinism:*** Nondeterminism is well-supported, this being STATECRUNCHER's speciality.
- ***(16) Priorities for transition execution:*** UML-style specialization priority (i.e. transitions on inner elements of the hierarchy) is currently implemented.
- ***(17) Pre-emptive versus non-pre-emptive interrupt:*** Not applicable, as it involves simultaneous events, whereas in STATECRUNCHER all user events are offered sequentially.
- ***(18) Distinguishing internal from external events:*** There is no formal distinction, except that a different PCO (point of control and observation) can be attached to each kind of event. Events that can be generated internally in an IUT are modelled by having them generated as fired events on the preceding transition in the STATECRUNCHER model, using nondeterministic constructs if the internal events only *may* occur.
- ***(19) Time specification, timeout, timed transition:*** No time support. Time handling is regarded as a test generator or test driver/harness affair (e.g. when we wish to wait for the SUT to perhaps execute an internal event). STATECRUNCHER

© Graham G. Thomason 2003-2004

can indicate that this is the situation by providing an event called e.g. `wait`, which has this special meaning.

- ***(Feature items - semantics, when not covered by the above)***
    - ***True concurrency***: No.
    - ***Discrete/continuous time***: Discrete
- ***(Feature items - syntax)***
    - ***Graphical/Textual***: Textual.
    - ***Negated trigger event***: No
    - ***Timeout event***: No
    - ***Timed transition item***: No
    - ***Disjunction of trigger events***: No
    - ***Trigger condition***: Yes
    - ***State reference:*** Yes
    - ***Assignment to a variable:*** Yes
    - ***Inter-level transition:*** Yes
    - ***History mechanism:*** Yes

Other statechart features that could be included in a comparison are (1-10 supported by STATECRUNCHER): (1) multiple target states, (2) orbital transitions, (3) traces, (4) nondeterministic worlds, (5) scoping operators, (6) points of control and observation, (7) upon enter and upon exit actions, (8) entering and exiting of states as internally generated events, (9) parameterised events, (10) a command language supporting: (i) output of transitionable or all events, (ii) re-instatement of previous worlds (iii) creation of new worlds, (iv) explicit killing of worlds, (v) implicit killing of worlds on trace violations, etc. Some features *not* currently supported by STATECRUNCHER: (A) lambda transitions (i.e. transitions on data values, not requiring events), (B) recursive state machine implantation.

[Zhang] **Fan Zhang and To-yat Cheung**

Optimal Transfer Trees and Distinguishing Trees for Testing Observable Nondeterministic Finite-State Machines

IEEE Transactions on Software Engineering, Vol. 29, No. 1, Jan. 2003

The approach is the *finite state machine* one, not the *labelled transition system* one. Testing a black box NFSM involves bringing it into a specific state, for which a transfer tree (TT) is required, and then verifying that it is in the correct state by further transitioning, using diagnosis/distinguishing trees (DTs). This paper investigates for *observable* NFSMs (different outputs generated on forks from the same event to different states) how, when weights (or probabilities) are assigned to nondeterministic transitions, TTs can be constructed to have a minimal expected value of weights over all paths, or to have minimal maximum of the weights. A similar problem for a certain kind of DT is also addressed.

5. Supporting projects / products / information

[Beveridge] **Jim Beveridge and Robert Wiener**
Multithreading Applications in Win32. The Complete Guide to Threads
Addison-Wesley, 1996, ISBN 0-201-44234-5 (Paperback)

A very good book on threads in Windows 32 systems. An example of using semaphores to protect against deadlock in the dining philosophers problem is given. (This problem is also considered by [Hoare], [Schneider] and many other textbooks on logic and parallelism).

[Boley] **Harold Boley**
Relationships between Logic Programming and XML
Proceedings of the 14th Workshop Logische Programmierung, Würzburg, Jan. 2000

The relevance of this paper is that it describes the nearest application of Prolog to a compiler-related field that we find in recent conferences on applications of Prolog (see [INAP 2001]), though for an early paper on the subject, see [Warren]. The paper shows how XML documents might be represented as PROLOG clauses and vice-versa, covering not just PROLOG *facts* but *relationships* with non-ground terms. The application to XML query languages is discussed, where a response can be that Prolog structures are nondeterministically enumerated.

[Bratko] **Ivan Bratko**
PROLOG Programming for Artificial Intelligence
Addison-Wesley, ISBN 0-201-41606-9

This book on PROLOG has an artificial intelligence slant. It is good on advanced tree structures and searching.

[Callahan] **John R. Callahan**
http://www.cs.wvu.edu/~callahan/interests.html

Callahan, and also the Nasa Goddard IV&V facility, (http://www.ivv.nasa.gov) interpret *verification* and *validation* in the following contexts:

© Graham G. Thomason 2003-2004

- Verification: Are we building the product right?
- Validation: Are we building the right product?

These are useful interpretations, corresponding to *testing* and *property checking*, but are by no means universally understood this way. Compare [IEEE 610.12.1990] and [CMMI].

[Clocksin 84] **W. F. Clocksin & C. S. Mellish**
Programming in Prolog
Springer Verlag, 1981. ISBN 3-540-11046-1

This is a standard Prolog book, using Edinburgh syntax. It is very well structured, and it clearly explains all constructs of the language with elementary examples.

[CMMI] **CMMI-SE/SW, Version 0.2b, Sept 1999**
Capability Maturity Model - Integrated Systems/Software Engineering
CMMI website: http://www.sei.cmu.edu/cmmi/cmmi.html

We seek definitions of *validation* and *verification*, and find:

- Validation (v.2, p.109): *The purpose of validation is to confirm that a product fulfills its intended use when placed in its intended environment.*
- Product Verification (v.2, p.106): *The purpose of Product Verification is to assure that work products meet the specified requirements*

The distinction between *property checking* and *implementation testing* does not appear to be made in these definitions. But see [Callahan] for a useful distinction.

[CYGWIN] **www.cygwin.com**

CYGWIN is a public-domain Linux-like environment for Windows. It consists of two parts: (1) a DLL (cygwin1.dll) which acts as a Linux emulation layer providing substantial Linux API functionality; (2) A collection of tools, which provide Linux look and feel. CYGWIN provides a platform for the popular test harness [DejaGnu].

[Darnell] **Peter A. Darnell and Philip E. Margolis**
C: A Software Engineering Approach
Springer-Verlag 2nd edition, 1988. ISBN 0-387-97389-3 / 3-540-97389-3

The ANSI C railroad syntax diagrams in this standard 'C' textbook give the basis of the expression grammar of STATECRUNCHER. In STATECRUNCHER an extension was used, and the left-recursive diagrams were transformed into a non-left recursive feed-forward grammar for parsing as a PROLOG DCG (Definite Clause Grammar), as described in [StCrGP4].

[DejaGnu]

This is an example of a public domain test harness, originally developed for Unix, using [TCL] (Tool Command Language) and [EXPECT]. It spawns a program (or several) and works by sending lines of input to its standard input, and receives standard output. It tests for a pattern match on the standard output or registers a timeout. *Pass* or *Fail* is logged per test, typically according to the success or failure of a pattern match. Philips has used it for state-based testing using state relation tables, from which tests are generated using a program written in TCL, effectively sending events and receiving the target states, matching against the tabular oracle. See [Savoye] for the manual.

[EXPECT]

Expect is a very powerful scripting language, built on [TCL], capable of spawning many processes and communicating with them independently via standard input and standard output. It is the underlying layer of the test harness [DejaGnu]. It is also useful for writing glue code in chains of testing tools, e.g. for converting one format or protocol to another, and is used as such in the integration of STATECRUNCHER into the [TorX] tool chain. The book on the language, written by its creator, is [Libes].

[IEEE 610.12.1990] **IEEE Standards, Software Engineering**
Volume I, Customer and Terminology Standards, 1999 Edition

We seek definitions of *validation* and *verification*, and find:

- Validation (p.80): *The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.*
- Verification (p.81): *(1) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (2) Formal proof of program correctness.*

The distinction between *property checking* and *implementation testing* does not appear to be made in these definitions. But see [Callahan] for a useful distinction.

[Libes] **Don Libes**
Exploring Expect
O'Reilley & Associates, 1995, ISBN 1-56592-090-2

The book by the creator of [EXPECT] describing [TCL] and EXPECT.

[INAP 2001] **The 14th International Conference of Applications of Prolog**
INAP 2001, held in Tokyo, 20-22 October, 2001.
http://www.ifcomputer.com/inap/inap2001/home_en.html

 © Graham G. Thomason 2003-2004

We examine the programme of this conference (and some previous years) to see what PROLOG is being used for, and whether it has been used as a compiler for what might be called a *domain specific language*, whether in the testing domain or any other. The session streams at this conference were:

- Supporting Organisational Learning: Knowledge Management and Case-based Reasoning
- Deductive Databases and Knowledge Management
- Web Applications for the Legal Domain
- Logic Programming for Natural Language Processing
- Practical Applications of Controlled Natural Languages
- Optimization and Simulation of Complex Industrial Systems. Extensions and Applications of Constraint-Logic Programming
- Business Opportunities in Advanced Technologies
- Decision Support in Medicine and Health Care
- Rule-Based Data Mining

Invited talks were on *Making decisions with incomplete information* (Donald Nute) and *The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations,* (Harold Boley). The latter is perhaps as close to the compiler domain as anything presented. For this subject area, see the related article [Boley]. For an article on the use of PROLOG for compilation, see [Warren].

[Koala] **R. van Ommering, F. van der Linden, J. Kramer, J. Magee**
The Koala Component model for Consumer Electronics Software
IEEE Computer, March 2000, pp. 78-85.

Koala is a static-binding component model, used for Philips TV software. The initial trialling of STATECRUNCHER is with Koala components and compositions of them.

[McCabe] **http://www.mccabe.com/main.htm**

The McCabe toolset provides for

- visualisation of code (C, C++ etc.), showing e.g. a module statement flow structure and, on a larger scale, what calls what.
- instrumentation of code, so that when tests are run, the degree of statement of branch coverage can be examined per module. The visualisation features show which statements were executed and which not. This is useful to reveal the effectiveness of (state-based) testing. See [Baker 95] for some Philips experience in this area.

[O'Keefe] **Richard O'Keefe**
The Craft of Prolog
MIT Press. ISBN 0-262-15039-5

A good PROLOG book with a particularly good section on the PROLOG ‘cut’.

[Ousterhout] **TCL and the TCL Toolkit**
John K Ousterhout
Addison Wesley, ISBN 0-201-63337-X

The above book is by the creator of [TCL] (Tool Command Language). TCL is a powerful scripting language, underlying [EXPECT] and the [DejaGnu] test harness.

[Savoye] **R. Savoye**
The DejaGnu Testing Framework
The Free Software Foundation, 1993

This is a manual for the [DejaGnu] public domain test harness.

[Sterling] **Leon Sterling & Ehud Shapiro**
The Art of Prolog
MIT Press, 1986. ISBN 0-262-19338-8

A good PROLOG book with many detailed examples, and useful guidance on good PROLOG programming style.

[SWI-Prolog] **http://www.swi.psy.uva.nl/projects/SWI-Prolog/**

A public domain PROLOG, used in addition to [WinProlog] for developing STATECRUNCHER.

[Tau] **www.telelogic.com**

A commercial tool by Telelogic for [TTCN] testing, with support for e.g. TCP/IP, RS-232 and "almost any target operating system"

[TCL] **Tool Command Language**

TCL is a powerful scripting language, underlying [EXPECT] and the [DejaGnu] test harness. It is described in [Ousterhout] and [Libes].

[TTCN] **The Tree and Tabular Combined Notation**
ISO (the International Organisation for Standardisation) /
IEC (International Electrotechnical Commission) standard 9646-3

A format and methodology for describing conformance tests, designed especially in connection with telecommunications standards and OSI protocols. Batch-generated state-based tests can be represented in TTCN. The basic structure is a depth first tree of alternatives (so supporting nondeterminism). A tutorial is available on the web by Mazen Malek

http://www.item.ntnu.no/~malek/research/TTCNcourse

 © Graham G. Thomason 2003-2004

[Warren] **David H.D. Warren**
Logic Programming and Compiler Writing
Software Practice and Experience, Vol. 10, 97-125 (1980).

This paper showed the feasibility of using PROLOG as an implementation language for compilers at an early date. The principle is illustrated for a 'toy' assembler, but the most important techniques are covered, including expression parsing with two operator precedences. The DCG (Definite Clause Grammar) technique is used, but without the more compact notation (the `->` operator, which hides systematically repeated parameters) which was later introduced into the PROLOG language. Computer memory and speed were restricting factors at the time; Warren considered memory the greater limitation. For STATECRUNCHER, a few megabytes of memory are needed, and speed is perhaps a limitation on PC machines below 300 MHz, corresponding to pre-1998 manufacture.

[WinProlog] **WinProlog, Logic Programming Associates Ltd**
http://www.lpa.co.uk

This is a version of PROLOG which was used for the development of STATECRUNCHER, on a PC (in addition to SWI-Prolog).

[WinRun] **WinRunner v4.0/v5.01, Mercury Interactive**
http://www.merc-int.com/products/winrunguide.html

A tool for Graphical-User-Interface-based testing of Window products. Philips has an extension, informally known as Deja Gnu-Y-Trewl, [Trew 98], to support state-relation tables. Another Philips tool that is useful in conjunction with WinRunner is GFET [GFET], which gives a graphical user interface to software that otherwise does not have one.

© Graham G. Thomason 2003-2004

6. STATECRUNCHER references

STATECRUNCHER documentation and papers by the present author

Main Thesis	[StCrMain]	The Design and Construction of a State Machine System that Handles Nondeterminism

Appendices

Appendix 1	[StCrContext]	Software Testing in Context
Appendix 2	[StCrSemComp]	A Semantic Comparison of STATECRUNCHER and Process Algebras
Appendix 3	[StCrOutput]	A Quick Reference of STATECRUNCHER's Output Format
Appendix 4	[StCrDistArb]	Distributed Arbiter Modelling in CCS and STATECRUNCHER - A Comparison
Appendix 5	[StCrNim]	The Game of Nim in Z and STATECRUNCHER
Appendix 6	[StCrBiblRef]	Bibliography and References

Related reports

Related report 1	[StCrPrimer]	STATECRUNCHER-to-Primer Protocol
Related report 2	[StCrManual]	STATECRUNCHER User Manual
Related report 3	[StCrGP4]	GP4 - The Generic Prolog Parsing and Prototyping Package *(underlies the STATECRUNCHER compiler)*
Related report 4	[StCrParsing]	STATECRUNCHER Parsing
Related report 5	[StCrTest]	STATECRUNCHER Test Models
Related report 6	[StCrFunMod]	State-based Modelling of Functions and Pump Engines

© Graham G. Thomason 2003-2004