Completing and Adapting Models of Biological Processes

Tiziana Margaria[1], Michael G. Hinchey[2], Harald Raffelt[3], James L. Rash[2], Christopher A. Rouff[4], and Bernhard Steffen[3]

[1] Chair of Service and Software Engineering, Universität Potsdam (Germany), margaria@cs.uni-potsdam.de
[2] NASA Goddard Space Flight Center, Information Systems Division, Greenbelt, MD, USA, michael.g.hinchey, james.l.rash@nasa.gov
[3] Chair of Programming Systems, Universität Dortmund (Germany), harald.raffelt,steffen@cs.uni-dortmund.de
[4] SAIC, Advanced Concepts Business Unit, McLean, VA 22102 rouffc@saic.com

Abstract. We present a learning-based method for model completion and adaptation, which is based on the combination of two approaches: 1) R2D2C, a technique for mechanically transforming system requirements via provably equivalent models to running code, and 2) automata learning-based model extrapolation. The intended impact of this new combination is to make model completion and adaptation accessible to experts of the field, like biologists or engineers. The principle is briefly illustrated by generating models of biological procedures concerning gene activities in the production of proteins, although the main application is going to concern autonomic systems for space exploration.

1 Motivation

A formal approach to Requirements-Based Programming, provisionally named R2D2C ("Requirements to Design to Code"), was developed at NASA [1] as a general-purpose method to mechanically transform system requirements into a provably equivalent model. This is a central need for ultra-high dependability systems like those developed at NASA for space exploration. The R2D2C approach provides mathematically tractable round-trip engineering for system development, rigorously based on formal modelling and formal reasoning techniques. In this paper we complement this method with a learning-based method for model completion and adaptation in order to make model completion and adaptation accessible to experts of the field, like biologists or engineers.

Before discussing the technical background and the biological application, we briefly sketch the standard areas of application.

Application Areas The work described below is motivated by the need for requirements-based programming for *ultra-high dependability* systems which are *remote*, *embedded*, and increasingly *autonomic*.

Sensor Networks An example of a sensor network for solar system exploration is the Autonomous Nano Technology Swarm mission (ANTS) [2], which is at the concept development phase. This mission will send 1,000 pico-class (approximately 1 kg) spacecraft to explore the asteroid belt. The ANTS spacecraft will act as a sensor network making observations of asteroids and analyzing their composition. Embedded sensors in space applications are a challenge along several research dimensions: large signal propagation delays in communications with Earth; unavailable or blocked communications paths between the spacecraft and mission control on Earth for variable (perhaps long) intervals of time; and operations under extremes of dynamic environmental conditions.

Due to the complexity of these systems as well as their distributed and parallel nature, they will have an extremely large state space and will be impossible to test completely using traditional testing techniques. R2D2C helps by converting the scenarios into a formal model that can be analyzed for concurrency-related errors, consistency and completeness, as well as domain-specific errors.

Robotic Operations We have been experimenting with generating code to control robots, but more interesting is the use of this approach to investigate the validity and correctness of procedures for complex robotic assembly or repair tasks in space, which rely heavily on the support of embedded controllers. Exploratory work here concerns providing an additional means to validate procedures from the Hubble Robotic Servicing Mission (HRSM) – for example, the procedures for replacement of cameras on the Hubble Space Telescope (HST).

Communication Systems The learning based approaches have fared quite promisingly for the test-based discovery of models of legacy communication systems, thus outperforming prior approaches based on trace combination [3]. As shown in [4, 5], the test-based model generation by classical automata learning is very expensive. It requires an impractically large number of queries to the system, each of which must be implemented as a system-level test case. Key towards the tractability of observation based model generation are powerful optimizations exploiting different kinds of expert knowledge in order to drastically reduce the number of required queries, and thus the testing effort. Recent studies have brought to a thorough experimental analysis of the second-order effects between such optimizations in order to maximize their combined impact [5], and to the development of a mature toolset for experimentation [6], which is used here. As shown in [7], our learning method is coherent with the usual notions of conformance testing.

In the specific R2D2C context, we investigate the possible application of the combined approach to the specification of communication mechanisms described in the previous application domains. This can be completed by a test-based or monitoring-based validation once those systems are operational.

In the following, we sketch the principles on which the R2D2C approach works and the effects of the learning-enhanced method.

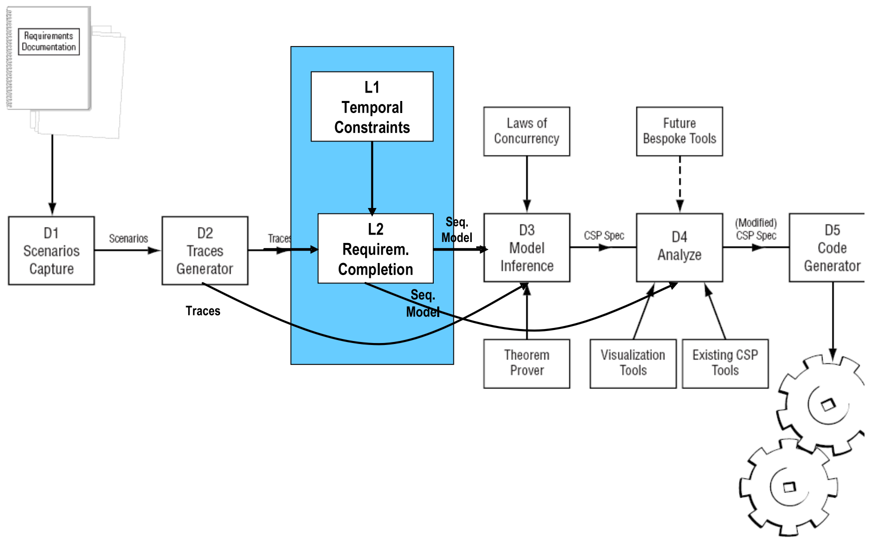

Fig. 1. The enhanced R2D2C Approach with Requirement Completion

2 How R2D2C Works

The R2D2C approach involves a number of phases, which are reflected in the system architecture described in Figure 1 and described below.

D1 Scenarios Capture: Engineers, end users, and others write scenarios describing intended system operation. The input scenarios may be represented in a constrained natural language using a syntax-directed editor, or may be represented in other textual or graphical forms.

D2 Traces Generation: Traces and sequences of atomic events are derived from the scenarios defined in D1.

D3 Model Inference: A formal model, or formal specification, expressed in CSP is inferred by an automatic theorem prover – in this case, ACL2 [8] – using the traces derived in phase 2. A deep[1] embedding of the laws of concurrency [9] in the theorem prover gives it sufficient knowledge of concurrency and of CSP to perform the inference. The embedding will be the topic of a future paper.

D4 Analysis: Based on the formal model, various analyses can be performed, using currently available commercial or public domain tools, and specialized tools that are planned for development. Because of the nature of CSP, the model may be analyzed at different levels of abstraction using a variety of possible implementation environments. This will be the subject of a future paper.

[1] "Deep" in the sense that the embedding is semantic rather than merely syntactic.

D5 Code Generation: The techniques of automatic code generation from a suitable model are reasonably well understood. The present modeling approach is suitable for the application of existing code generation techniques, whether using a tool specifically developed for the purpose, or existing tools such as FDR [10], or converting to other notations suitable for code generation (e.g., converting CSP to B [11] and then using the code generating capabilities of the B Toolkit).

According to this full cycle, developing a system that will have a high level of reliability requires the developer to represent the system as a formal model that can be proven to be correct. Through the use of currently available tools, the model can then be automatically transformed into code with minimal or no human intervention to reduce the chance of inadvertent insertion of errors by developers. Automatically producing the formal model from customer requirements would further reduce the chance of human error insertion.

In this paper we focus on a specific, new aspect of the R2D2C approach, the completion of the requirements given as a set of traces as generated by D2. This needs a short introduction into automata learning.

3 Automata Learning

Machine learning deals in general with the problem how to automatically generate a system's description. Besides the synthesis of static soft- and hardware properties, in particular invariants [12], [13], [14], the field of *automata learning* is of particular interest for soft- and hardware engineering [15], [16], [17], [18], [19].

Automata learning tries to construct a deterministic finite automaton (see below) that matches the behavior of a given target automaton on the basis of observations of the target automaton and perhaps some further information on its internal structure. [3, 20, 21] explain our view on the use of learning. Here we only summarize the basic aspects of our realization, which is based on Angluin's learning algorithm L^* from [22].

L^*, also referred to as an *active* learning algorithm, learns a finite automaton by *actively* posing *membership* queries and *equivalence* queries to that automaton in order to extract behavioral information, and refining successively an own hypothesis automaton based on the answers. A membership query tests whether a string (a potential run) is contained in the target automaton's language (its set of runs), and an equivalence query compares the hypothesis automaton with the target automaton for language equivalence, in order to determine whether the learning procedure was (already) successfully completed and the experimentation can be terminated.

3.1 Learning-Based Model Completion and Adaptation

Specifications in terms of individual traces are by their nature very partial and represent only the most prominent situations. This partiality is one of the major problems in requirement engineering. It often causes errors in the system design that are difficult to fix. Thus techniques for systematically completing and later on adapting such partial requirement specifications in cooperation with the application expert are of major practical importance.

We therefore propose a method for requirements completion and adaptation, based on automatic (active) automata learning. In essence, the method

- *initializes* the learning algorithm with the set of traces constituting the requirement specification and with the model needing adaptation (this model may well be empty), and
- constructs a *consistent behavioral model* by establishing predefined consistency and well-foundedness conditions. The details of how to do this have been explained in [20] its practical handling in [4, 5], and a library-based toolset for experimentation in [6].

In this fashion, we arrive at a finite state behavioral model, which is an *extrapolation* of the given requirement specification: it comprises *all* 'positive' traces of the specification, and rejects all forbidden traces. All the other potential traces are consider as 'don't cares', in order to construct a corresponding state minimal hypothesis automaton. In particular, although the learning procedure by its nature will only investigate finitely many traces, the constructed hypothesis automaton will typically accept infinitely many traces, as the extrapolation process introduces loops.

For this method to work, a number of membership queries need to be answered. Both, establishing closure of the model, as well as establishing the consistency of the abstraction of reaching words into states (i.e., of the characterization from above introduced in the previous section) can only be effected on the basis of additional information about the intended/unknown system.

3.2 Requirement Completion in R2D2C

Fig. 1 shows the R2D2C scenario including the new requirement completion components. As indicated by the arrows representing the potential flow of R2D2C processes, our new components introduce the following new options, which complement the original R2D2C process here indicated by the arrow bypassing the requirements completion module L2:

- Most powerful is the integrated mode of use, where the requirement completion component L2 is added to the original process. Its role is here simply to support the evaluation of the given set of requirement traces, and to hint at underspecified portions which may be successively completed. This option strengthens the original R2D2C process.

– Alternatively, one may replace the model inference component D3 by our requirements completion component L2, meaning that the subsequent component D4 and D5 directly work on the model produced by L2. Currently, this means that we restrict ourselves to sequential models. However, we are investigating how to overcome this restriction in the future.

The next section presents a non-standard application of our technology to the description of biological processes.

4 Application: Generating and Verifying Complex Biological Scripts and Procedures

Finding patterns in biological sequences has the goal of identifying parts that have a biological meaning [23, 24, 25]. There are several approaches to this problem. Bioperl [26] provides a collection of perl modules used for the development of perl scripts for use in Bioinformatics applications.

The Bioperl [27, 28] Project is an international association of developers of open source Perl tools for bioinformatics, genomics and life science research, with strongly increasing relevance over the almost 10 years. Bioperl relies on a large number of scripts to access, steer, and orchestrate a growing number of bioinformatic tools and databases. These scripts are becoming increasingly complex and intertwined, so that their correctness has become a legitimate concern of the community.

The application of software validation techniques to Bioperl is attempting to provide an ongoing, systematic testing of the Bioperl basis, with patches and validated new code being added to the public codebase. The goal is to establish user confidence that software components will work as described. R2D2C is a comprehensive software validation method that has been already successfully applied to problems in this domain.

We consider here again the application example already handled with R2D2C in [29] and solve the model creation problem with the combined methodology, using the requirement completion in replace mode.

4.1 From Scenarios to CSP

Let us consider again the same example from [30] (pp. 146-147). The problem is described in the form of a scenario:

– Gene *GeneOne* produces protein *ProteinOne* in t1 units of time; *ProteinOne* dissipates in time u1 and triggers condition *cone*.
– Gene *GeneTwo* produces protein *ProteinTwo* in t2 units of time; *ProteinTwo* dissipates in time u2 and triggers condition *ctwo*.
– Once produced, *ProteinTwo* positions itself in *GeneOne* for u2 units of time preventing *ProteinOne* from being produced.

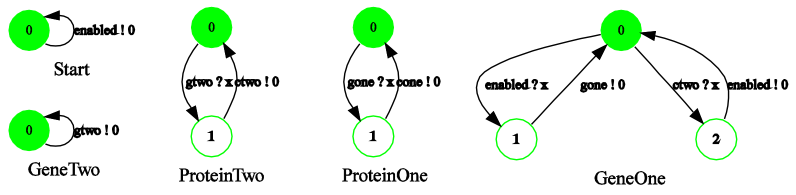

Fig. 2. Learned models of the single actors

The scenario represents a process that is expressed and implemented in Bioperl using a Perl script. However, it is also possible to express this scenario using a formal model based on CSP [31]. *GeneOne, ProteinOne, GeneTwo, ProteinTwo* can be considered as separate processes with timing constraints implicitly included. (Timing constraints may be explicitly handled by using Timed CSP, a variant of CSP which extends the semantics of CSP with time [32].) The implicit pre-condition that *GeneOne* must be enabled is handled by the *Start* process. The events and conditions describing protein production are represented as messages *gone, cone, gtwo, ctwo*, and *enabled*. The resulting R2D2C input scenario is (D1):

```
Start sends enabled.
GeneOne receives enabled then sends gone.
ProteinOne receives gone then sends cone.
GeneTwo sends gtwo.
ProteinTwo receives gtwo then sends ctwo.
GeneOne receives ctwo then sends enabled.
```

and the corresponding system description in CSP (after Phase D2):

```
channel cone, ctwo, enabled, gone, gtwo : T ;
Start = enabled ! 0 -> Start ;
GeneOne = enabled ? x -> gone ! 0 -> GeneOne ;
ProteinOne = gone ? x -> cone ! 0 -> ProteinOne ;
GeneTwo = gtwo ! 0 -> GeneTwo ;
ProteinTwo = gtwo ? x -> ctwo ! 0 -> ProteinTwo ;
GeneOne = ctwo ? x -> enabled ! 0 -> GeneOne ;
System =
     GeneOne [| {| |} |]
     GeneTwo [| {| |} |]
     ProteinOne [| {| |} |]
     ProteinTwo [| {| |} |]
     Start ;
```

4.2 Learning and Adapting the Models

Instead of analyzing the CSP model, as in [29], we have here used our learning technique to fully automatically produce automata models for each system component (see Fig. 2), as well as for the model of the whole system (Fig. 3).

These graphs show in a very intuitive way the global behaviour of the system. It is thus very direct also for someone unfamiliar with CSP and its tools to validate the behaviour by inspection.

A frequent mistake in implementing these requirements is in fact the omission of constraints, either due to their implicit presence in the requirements, or due to errors in code development. For example, omitting *Start* sends *enabled* (which makes explicit an implicit precedence) nothing prevents *GeneOne* from constantly generating *ProteinOne* and ignoring *ProteinTwo* inhibition. The corresponding erroneous system of [29] has also been learnt with our method, resulting in the global behaviour of Fig. 4(4).

This inspection could then be used to revise the requirements before developing the Bioperl code, even before carrying out a formal analysis at D4.

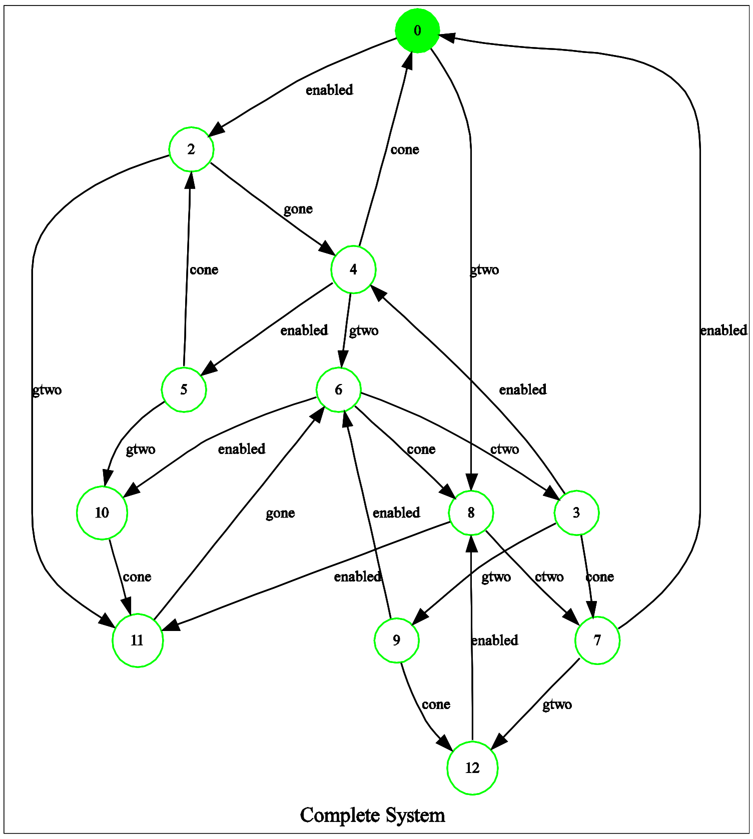

Fig. 3. Learned model of the biological system

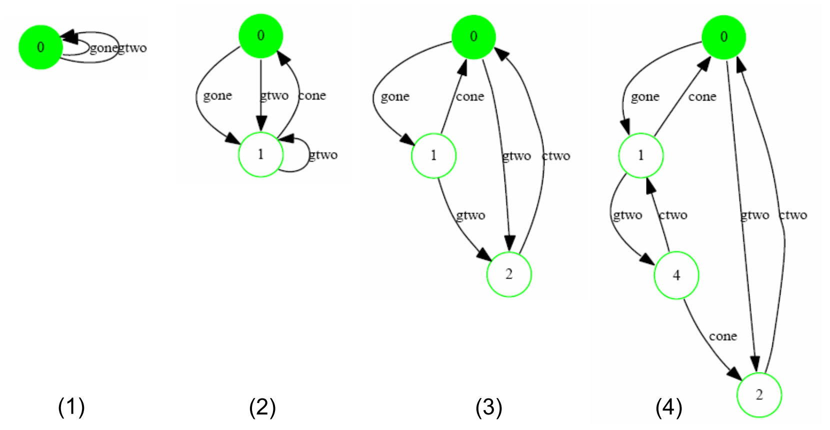

Fig. 4. Stepwise learned model of the incorrect biological system

4.3 Successive Refinement

The erroneous system could be learned in only four iterations. Fig. 4 illustrates the concrete learning process starting from the initial hypothesis along the application of the algorithm.

To learn this model from scratch we initialize the learning algorithm with no information about the system except for the alphabet of symbols. No initial trace is provided, no hints on possible symmetries or independent actions.

1. After processing the queries of length 0 and 1 with these outcomes

()	*acc*
gone	*acc*
gtwo	*acc*
ctwo	*nonacc*
cone	*nonacc*

   the learning algorithm generates the hypothesis model depicted in Figure 4(1): there is at least one state, which accepts *gone* and *gtwo* and rejects *cone* and *ctwo*. In the picture we show only the accepting traces: the automata are incomplete in the sense that all the absent symbols lead to a single nonaccepting state.
2. By model checking an expert-given corresponding property we find out that *gone.gone* is not an accepting sequence, thus the model (1) is not yet accurate and must be refined. We refine it starting from this counterexample, and reach a new hypothesis shown in Fig. 4(2). Here, the counterexample sequence leads to the discovery of a second state, state 1, and we have distinguished further behaviours.
3. Due to expert knowledge, we find out that *gtwo.gtwo* is another trace that must be rejected. This leads to the further refinement of state 1 and by completion we reach a new hypothesis as in Fig. 4(3).

4. After also rejecting *gone.gtwo.ctwo.gone* in a similar fashion, we arrive at the automaton shown in Fig. 4(4), which satisfies all our expectations.

In order for this method to scale, and to limit the required expert-interaction, we provide a number of optimizations that exploit other sources of expert knowledge, like prefix closure of the language, symmetry between certain components (genes always behave like genes), and the independence of certain observations.

5 Conclusions and Perspectives

We have presented a learning-based method for model completion and adaptation, which is based on the combination of two approaches: 1) R2D2C, a technique for mechanically transforming system requirements via provably equivalent models to running code, and 2) automata learning-based model extrapolation. The intended impact of this new combination is to make model completion and adaptation accessible to experts of the field, like biologists or engineers.

Currently, we are investigating the power of our method. Until now, we used it for an initial model completion, as a support for the creation of the first model. We are currently carrying out case studies that concern model evolution and change, in this case continuously updating the model of biological processes according to new information.

We are also building and adapting models of servicing procedures for spacecrafts, and adaptive control procedures for remote autonomic systems. These are the application areas that in our opinion are going to profit enormously of the combined completion-adaptation technique.

References

1. Michael G. Hinchey, James L. Rash, Christopher A. Rouff: *A Formal Approach to Requirements-Based Programming,* Proc. ECBS 2005, 12th IEEE Int. Conf. on the Engineering of Computer-Based Systems, Greenbelt (MD), 2005, IEEE, pp. 339–345.
2. S. A. Curtis, J. Mica, J. Nuth, G. Marr, M. L. Rilee, and M. K. Bhat. ANTS (Autonomous Nano-Technology Swarm): An artificial intelligence approach to Asteroid Belt resource exploration. In *Proc. Int'l Astronautical Federation, 51st Congress*, October 2000.
3. A. Hagerer, H. Hungar, O. Niese, and B. Steffen. Model Generation by Moderated Regular Extrapolation. *Proc. of the 5th Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2002)*, LNCS 2306, pp. 80–95.
4. H. Hungar, T. Margaria, B. Steffen: *Test-Based Model Generation for Legacy Systems*, IEEE International Test Conference (ITC), Charlotte, NC, September 30 - October 2, 2003.
5. T. Margaria, H. Raffelt, B. Steffen: *Analyzing Second-Order Effects Between Optimizations for System-Level Test-Based Model Generation*, Proc. IEEE International Test Conference (ITC), Austin, TX (USA), November 8 - 10, 2005, IEEE Computer Society Press.

6. H. Raffelt, B. Steffen, T. Berg: *LearnLib: A Library for Automata Learning and Experimentation*, Proc. FMICS 2005, 10th ACM Workshop on Formal Methods for Industrial Critical Systems, Lisbon, Sept. 2005.
7. T. Berg, O. Grinchtein, B. Jonsson, M. Leucker, H. Raffelt, B. Steffen: *On the Correspondence Between Conformance Testing and Regular Inference*, Proc. FASE 2005, 8th Int. Conf. on Fundamental Approaches to Software Engineering, Edinburgh, UK, April 2005, LNCS N.3442, pp. 175–189, Springer Verlag, 2005.
8. M. Kaufmann and Panagiotis Manolios and J Strother Moore. *Computer-Aided Reasoning: An Approach.* Advances in Formal Methods Series. Kluwer Academic Publishers, Boston, 2000.
9. M. G. Hinchey and S. A. Jarvis. *Concurrent Systems: Formal Development in CSP.* International Series in Software Engineering. McGraw-Hill International, London, UK, 1995.
10. *Failures-Divergences Refinement: User Manual and Tutorial.* Formal Systems (Europe), Ltd., 1999.
11. M. J. Butler. *csp2B : A Practical Approach To Combining CSP and B.* Declarative Systems and Software Engineering Group, Department of Electronics and Computer Science, University of Southampton, Feb. 1999.
12. M. D. Ernst, A. Czeisler, W. G. Griswold, D. Notkin. Quickly detecting relevant program invariants In proceedings of the 22nd *International Conference on Software Engineering* (ICSE 2000), 449–458, June 2000.
13. J. W. Nimmer, M. D. Ernst. Automatic generation of program specifications In Proceedings of the 2002 *International Symposium on Software Testing and Analysis* (ISSTA 2002), 232–242, July 2002
14. Y. Brun, M. D. Ernst. Finding latent code errors via machine learning over program executions Proc. 26th *Int. Conf. on Software Engineering* (ICSE'04), pp. 480–490, May 2004
15. J. E. Cook, A. L. Wolf Discovering Models of Software Processes from Event-Based Data ACM Trans. on Software Engineering and Methodology (TOSEM) pp. 215–249, 1998
16. L. Mariani, Mauro Pezzè. A technique for verifying component-based software Proceeding of the *Int. Workshop on Test and Analysis of Component Based Systems*, TACOS 2004, Barcelona, March 2004
17. T. Xie, D. Notkin Mutually Enhancing Test Generation and Specification Inference. In Proceedings of 3rd *International Workshop on Formal Approaches to Testing of Software* (FATES 2003), LNCS Vol. 2931, Springer, pp. 60–69, Oct. 2003.
18. D. Peled, M. Y. Vardi, M. Yannakakis Black Box Checking Formal Methods for Protocol Engineering and Distributed Systems, (FORTE/PSTV), pp. 225-240, 1999, Kluwer.
19. J. E. Cook, Z. Du, C. Liu, A. L. Wolf. Discovering Models of Behavior for Concurrent Systems Tech. rep. New Mexico State University, Dept. of Computer Science, Aug. 2002
20. B. Steffen and H. Hungar, Behavior-based model construction. In S. Mukhopadhyay and L. Zuck, editors, *Proc. 4th Int. Conf. on Verification, Model Checking and Abstract Interpretation*, LNCS 2575, Springer 2003.
21. T. Margaria, O. Niese, H. Raffelt, and B. Steffen Efficient Test-based Model Generationfor Legacy Reactive Systems. To appear in Proceedings of International High Level Design Validation and Test Workshop, 2004 Sonoma, California.

22. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 2(75):87–106, 1987.
23. D. E. Krane and M. L. Raymer. Fundamental Concepts of Bioinformatics. Benjamin Cummings, San Francisco, 2003.
24. S. A. Krawetz and D. D. Womble. Introduction to Bioinformatics: Theoretical and Practical Approach. Humana Press, Totowa, New Jersey, 2003.
25. A. M. Lesk. Introduction to Bioinformatics. Oxford University Press, Oxford, UK, 2002.
26. J. Stajich and E. Birney. The Bioperl project: motivation and usage. SIGBIO Newsl., 20(2):1314, 2000.
27. P. van Heusdan. Applying software validation techniques to Bioperl. In 2004 Bioinformatics Open Source Conference, Glasgow, UK, 2930 July 2004. Abstract.
28. Stajich JE, Block D, Boulez K, Brenner SE, Chervitz SA, Dagdigian C, Fuellen G, Gilbert JG, Korf I, Lapp H, Lehvaslaiho H, Matsalla C, Mungall CJ, Osborne BI, Pocock MR, Schattner P, Senger M, Stein LD, Stupka E, Wilkinson MD, and Birney E: *The Bioperl toolkit: Perl modules for the life sciences.* Genome Res 2002 Oct; 12(10) 1611-8. PubMed HubMed [bioperl2002]
29. J. Rash, M. Hinchey, D. Gracanin, C. Rouff: *An Approach to Generating and Verifying Complex Scripts and Procedures*, 4th IEEE-CS Computational Systems Bioinformatics, CSB Workshops, Stanford, Aug. 2005, pp. 305–313.
30. J. Cohen. Bioinformaticsan introduction for computer scientists. ACM Comput. Surv., 36(2):122158, 2004.
31. C. A. R. Hoare. Communicating Sequential Processes. Prentice Hall International Series in Computer Science. Prentice Hall International, Englewood Cliffs, NJ, 1985.
32. S. Schneider, J. Davies, D. M. Jackson, G. M. Reed, J. Reed, and A. W. Roscoe. Timed CSP: Theory and practice. In Proc. REX, Real-Time: Theory in Practice Workshop, volume 600 of LNCS, pages 640-675. Springer-Verlag, 3–7 June 1991.