Rapid Prototyping Applications of Formal Reasoning Tools to Biological Cellular Signalling Networks

By
Romulus Apolzan

A thesis submitted for the degree of
Bachelor of Computing Science (Honours)
of The University of New England
December 2005

DECLARATION

I certify that the substance of this thesis has not already been submitted for any degree and is not currently being submitted for any other degree.

I certify that to the best of my knowledge, any help received in preparing this thesis, and all sources used, have been acknowledged in this thesis.

. . . .

Acknowledgements

I would like to thank my supervisor Ian Mason for his invaluable help, patience and guidance during this Honours project. The many afternoons spent solving my programs' bugs are highly appreciated.

I would like to thank Carolyn Talcott for her generosity, enthusiasm and constant support during the development of this work. I would also like to thank Merrill Knapp for the useful explanations.

Last but not least I would like to thank my family for their support and encouragment.

Abstract

In this thesis biological cellular signalling networks are modelled and interrogated. We make use of two state-of-the-art formal reasoning tools: Symbolic Analysis Laboratory (SAL) and NuSMV. Using Perl scripts, we generate SAL and NuSMV specifications that model the cellular networks. We do this by translating a specification produced from curated Maude models.

Throughout our experiments we use Temporal Logic queries to compare the efficiency of generating cellular execution paths. We propose several rules of thumb, that should be used as a guide when deciding which of the SAL or NuSMV tools would be best to use to answer a given query if integrated into the Pathway Logic Assistant (PLA). PLA is a tool that supports visualization and interaction with the biological models. The underlying infrastructure for the PLA is the Interoperability Platform (IOP), a system that allows tools to interact and interoperate.

We also implement and integrate in IOP two classes of SAL actors. The intended outcome is to make communication possible between Maude and SAL. Currently, any IOP user can communicate with SAL via the SAL actors.

We explain the difficulties and reasons for implementing two classes of SAL actors, showing that the tools within SAL fit into two categories: the first category contains programs that terminate after executing a request, and the second category contains systems programmed as a “read-eval-print” loop.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis presents the results of modelling and querying biological cellular signalling networks, using formal reasoning tools, such as SAL [16], NuSMV [13], Maude [12] and Lola [1]. The intended outcome of this work is to compare the performances of these tools, and to find well argued rules of thumb that should be used as a guide when deciding which of the SAL or NuSMV tools would be best to use to answer a given query if integrated into the Pathway Logic Assistant [14].

We also implement and integrate in the Interoperability Platform [10] two classes of SAL actors.The intended outcome is to make communication possible between SAL and other formal reasoning tools, such as Maude and PVS [15], which are already amongst the actors available in the IOP platform. We explain the difficulties and reasons for implementing two classes of actors, by indentifying two categories of SAL tools: the first category contains programs that terminate after executing a request, and the second category contains systems programmed as a "read-eval-print" loop.

Chapter 2 gives an overview of formal methods and presents two formal reasoning tools of interest: SAL and NuSMV. We explain which parts of SAL and NuSMV are used in our work (i.e. sal-smc, sal-bmc, NuSMV -bmc).

Chapter 3 outlines the main concepts involved in Linear Temporal Logic and Computation Tree Logic.

Chapter 4 presents the main features of symbolic model checking and bounded model checking. We present the type of properties that can be verified, using state-of-the-art model checkers, such as SAL and NuSMV.

Chapter 5 presents the basic concepts involved in Pathway Logic. Our experiments are largely based on this approach. Thus, we present how Pathway Logic is based on a powerful logic called "Rewriting Logic" implemented in Maude. Using Pathway Logic, biological models can be described at very different levels of abstraction, dynamic pathways can be generated using search and model checking techniques, and Maude models can be transformed into Petri nets for analysis and visualization.

Chapter 6 presents the subject of our experiments: biological cellular signalling networks. We give an overview of fundamental biological concepts such as signal transduction, naming conventions, activation of biological elements and biological networks used in our work. We also present the types of biological queries used in our experiments, expressed in Linear Temporal Logic and Computation Tree Logic.

Chapter 7 contains the core results of our work and compares the performances of the formal reasoning tools used. We also describe the concept of Dish, used throughout our experiments.

Appropriate conclusions are drawn in chapter 8 and rules of thumb are formulated. Further improvements and research directions are suggested.

Appendix A contains the core results of our experiments in more detail.

Appendix B presents the implementation details and integration of two classes of SAL actors into the Interoperability Platform. We present a sample use of these actors.

Chapter 2

Formal Methods Tools

2.1 Introduction

Formal methods are mathematically based techniques used throughout the development of a system to capture and define system and functional requirements [3]. Some formal methods are based on set theory and first order predicate calculus. Others are based on temporal logic, which is an extension of propositional logic to formalize how the truth values of some propositions alter with the time at which they are evaluated.
Our main interest is to use formal methods based on Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) in order to analyze the executable models of biological networks.

2.2 SRI tools

The Computer Science Laboratory from SRI [5] has a long tradition of over 30 years of building tools to support formal methods. Currently, their most widely used tools are [37]:

- The PVS verification system [15] that provides a specification language in which mathematical theories and conjectures about them can be specified, together with an interactive theorem prover that is used to discharge the conjectures.

- The Symbolic Analysis Laboratory (SAL) model checking toolkit [16] that is dedicated to the analysis of computational systems specified as transition relations.

- The Integrated Canonizer/Solver (ICS) decision procedures toolkit [9] that provides the deductive core for bounded model checking and automated k-induction in the SAL toolkit.
- Maude [12], a reflective language and system, that supports logical reflection and is used to create executable environments for different logics, theorem provers, languages, and models of computation.

Since SAL is designed for the modular specification of nondeterministic state machines by means of transition relations, we chose it as a tool for modelling and querying biological networks.

2.3 SAL's features

SAL provides a language similar to PVS, but specialized for the specification of state machines [33]. The core of the SAL language supports the specification of transition relations or nondeterministic state machines, either as guarded commands or as invariant definitions, or a combination of the two. The transition relations can be packaged in parameterized modules that can be composed synchronously or asynchronously.

The current SAL toolset provides explicit state, symbolic, bounded, infinite bounded and witness model checkers. We are interested in querying biological networks, using LTL and CTL properties. The state-of-the-art performance in checking LTL and CTL queries is provided by the symbolic, bounded and witness model checker, as stated in [37]. Below, we give a short description of the most important SAL tools.

2.3.1 SAL - Symbolic Model Checker (SAL-SMC)

SAL-SMC allows the specification of properties in Linear Temporal Logic. LTL formulas state properties about each linear path induced by a transition system or module. SAL-SMC tries to prove valid LTL properties and to generate counterexamples for invalid properties. SAL-SMC uses ICS as its default satisfiability (SAT) solver for optimising and delivering performance comparable to other state-of-the-art symbolic model checkers, like NuSMV [13].

2.3.2 SAL - Bounded Model Checker (SAL-BMC)

SAL-BMC also uses Linear Temporal Logic as its assertion language and specializes in finding finite counterexamples, no longer than some specified depth. By default, SAL-BMC uses ICS as its SAT solver, but can be optionally instructed to use other SAT solvers, like zChaff [20], BerkMin [6] or Siege [17]. We exploited this capability when we had to reduce the execution time and compare different LTL queries on complex networks.

2.3.3 SAL-INF-BMC and SAL-WMC

The infinite bounded model checker SAL-INF-BMC specializes in finding symbolic counterexamples, like its finite counterpart SAL-BMC. It provides a mechanism of checking infinite state systems defined over reals, integers, propositional calculus and propositional sets[33]. SAL-INF-BMC uses ICS [9] as its default SAT solver. It can be instructed to use other SAT solvers like UCLID [19], SVC [18] , CVC [7] or CVC-Lite [8], but with restrictions and without generating counterexamples.

There are two reasons that we have used SAL-BMC rather than SAL-INF-BMC in our experiments. Firstly, SAL-INF-BMC cannot reduce its computation time efficiently, when generating counterexamples for invalid LTL properties. Secondly, the biological models analyzed in our experiments are in fact finite state models.

The witness model checker SAL-WMC implements a novel approach that constructs both symbolic witnesses (for true properties) and counterexamples (for false properties) for assertions in full CTL [37]. However, SAL-WMC never worked successfully in our experiments.

2.3.4 Tool selection

It is worth mentioning that SAL-SMC and SAL-BMC proved to be SAL's most efficient tools in our experiments. Thus, our decision was to concentrate our experiments on using the symbolic model checker and the bounded model checker from the SAL toolset. These two model checkers were sometimes able to deal even with CTL queries, as shown in our experiments.

2.4 NuSMV platform

NuSMV is a robust and flexible platform for symbolic model checking [13], developed as a joint project between Carnegie Mellon University (CMU) and Istituto per la Ricerca Scientifica e Tecnologica (IRST). Like SAL, NuSMV is designed for the modular specification of nondeterministic state machines by means of transition relations, declaration and instantiation mechanisms for modules and processes, corresponding to synchronous and asynchronous composition.

We have used NuSMV in our experiments to model and query biological networks, using LTL properties. Below, we give a short description of the most important NuSMV tool, used in our experiments, and presented in [30].

2.4.1 Bounded Model Checker

This tool provides the SAT-based model checking functionalities. Like SAL-BMC, the bounded model checker within NuSMV (i.e. NuSMV -bmc) also uses Linear Temporal Logic as its assertion language and is specialized in finding finite counterexamples, no longer than some specified depth. By default, NuSMV -bmc uses SIM as its SAT solver, but can be optionally instructed to use other SAT solvers like zChaff [20]. SIM is based on the Davis-Logemann-Loveland procedure, and proved to be an efficient SAT solver in our experiments.

Since the experiments in this research thesis are largely based on the Temporal Logic framework, we provide a basic description of the main concepts involved in Temporal Logic.

Chapter 3

Temporal Logic

3.1 Introduction

Temporal logic is one of the largest and most active areas of philosophic logic. It was introduced around 1960 by Arthur Prior under the name of Tense Logic. Temporal logic is used to describe any system of states and rules, and reason about propositions qualified in terms of time [46]. If predicate logic allows reasoning about a state, temporal logic allows reasoning about sequences of states.

3.2 Background

The semantics of temporal logic is defined in terms of *Kripke structures* [4].

Definition 1 *A Kripke structure is a tuple $M :=< S, R, S_0, A, P >$ where:*

- *S is a set of states*
- *$R \subseteq S \times S$ is a transition relation*
- *$S_0 \subseteq S$ is a set of initial states*
- *A is a set of atomic propositions.*
- *$P : S \rightarrow \rho(A)$ labels each state with the set of atomic propositions satisfied by the state.*

A Kripke structure is also known as a *labelled state transition graph.*

Given some atomic formulas A, B, ..., our interest is to formulate new propositions about A, B, ..., that will hold at some points of time in the future. Basic temporal operators were introduced to handle this situations [2]:

- The **next time operator** $\bigcirc A$ states that *A holds at the next point of time in the future.*
- The **always/henceforth operator** $\Box A$ states that *A holds at all coming points of time in the future.*
- The **eventually/sometime operator** $\Diamond A$ states that *A holds at some point of time in the future.*

If a state $s_n \in S$ satisfies a formula A at some point in the future, we are interested in building a path that leads to s_n via a sequence of states according to the transition relation.Therefore, we use the definition stated in [4], to define a path π in a Kripke structure M:

Definition 2 *A path in a Kripke structure M is an infinite sequence of states* $\pi := s_0 s_1 \ldots \in S$ *such that* $s_0 \in S_0$ *and for all* $i \geq 0$*, we have* $(s_i, s_{i+1}) \in R$ *in which case we write* $s_i \rightarrow s_{i+1}$.

3.3 Linear Temporal Logic (LTL) : Syntax and Semantics

Linear Temporal Logic reflects a linear and discrete nature of time. Each point in time has in LTL one unique possible future. LTL has the following temporal operators:

- Gp is the equivalent of the *always operator* $\Box p$, and states that it is always going to be the case that p holds.
- Fp is the equivalent of the *eventually operator* $\Diamond p$, and states that it will be the case at some state in the future that p holds.
- Xp is the equivalent of the *next time operator* $\bigcirc p$, and states that in the immediately succeeding state it will be the case that p holds.

- $p\ U\ q$ or the *until operator* states that p will hold until at some point in the future q will hold.

We use the definition stated in [4] to specify when a path beginning at state s satisfies a given formula:

Definition 3 *We say that a state s of a Kripke structure $M :=< S, R, S_0, A, P >$ satisfies a formula p, written $s \models p$ iff for every path π in M, starting at s, we have $\pi \models p$.*

These kind of paths that satisfy a given formula or LTL property p are extremely valuable because it enables us to check whether:

- A formula p will eventually be true on a path π, i.e. Fp or
- A formula p will always be true on a path π, i.e. Gp.

Therefore, we use the definition stated in [29] to specify when a formula p is true along a path π, in a Kripke structure M. The clauses for ordinary boolean connectives are omitted. We denote by π^k the k-shifted suffix of π, that is $\pi^k := s_k s_{k+1} \ldots$.

Definition 4 *Given a Kripke structure M, the relation $\pi \models p$, which states that a formula p holds in the path π, is inductively defined as follows:*

- *$\pi \models p$ iff $s_0 \models p$, where s_0 is the starting state of π,*
- *$\pi \models Xp$ iff $\pi^1 \models p$,*
- *$\pi \models Fp$ iff there exists $k \geq 0$ such that $\pi^k \models p$,*
- *$\pi \models Gp$ iff for every $k \geq 0$, $\pi^k \models p$,*
- *$\pi \models pUq$, where q is another formula, iff there exists $k \geq 0$ such that $\pi^k \models q$ and $\pi^j \models p$ for all $0 \leq j < k$.*

3.4 Computation Tree Logic (CTL) : Syntax and Semantics

Computation Tree Logic was first introduced by Emerson and Clarke [31]. CTL differs from LTL in being a branching time logic that allows each time-point to split into a variety of possible futures. Hence, LTL concentrates on properties with a single execution path $\pi = s_0 s_1 s_2 \ldots$, while CTL allows quantification over the paths or branches of the computation tree by using the following operators:

- AGp states that p is globally true along every path.
- EGp states that p is true for all states along at least one path.
- AFp states that along every path there is at least one state in which p holds.
- EFp states that along at least one path there is at least one state in which p holds.
- AXp states that for all immediately succeeding states of the current state, p holds.
- EXp states that there is a successor state where p holds.
- $A(pUq)$ states that for all paths, p holds until a state is reached where q holds.
- $E(pUq)$ states that there is at least one path where p holds until a state is reached where q holds.

Like in LTL, CTL formulas are evaluated against Kripke structures. The main difference between LTL and CTL, observed in [52], is that LTL is a path-based framework whereas CTL is tree-based. This implies the following:

- in LTL, linear time formulas make assertions about a path and when being evaluated are true or false with respect to this particular path.
- CTL or branching time formulas make assertions about the states of a system so that their truth values depend on a particular state.

A consequence of these facts is that LTL and CTL formulas coincide if the model considered has only one path. However, CTL formulas like EF(p) which states that "p potentially holds" are not expressible in LTL. On the other hand, strong fairness properties such as "every process enabled infinitely often is also executed infinitely often" [38], can be expressed in LTL as: GF(*enabled*) => GF(*executed*), but cannot be expressed in CTL. The most useful formulas are expressible in both frameworks:

- Safety properties such as "p will always hold": LTL – G(p), CTL – AG(p).

- Liveness properties such as "action q must follow action p":
  LTL – G(p ⇒ F(q)), CTL – AG(p ⇒ AF(q)).

The complex semantics of CTL formulas can be understood more clearly, if they are represented using computation trees, as Clarke et al. have proposed in [31]:

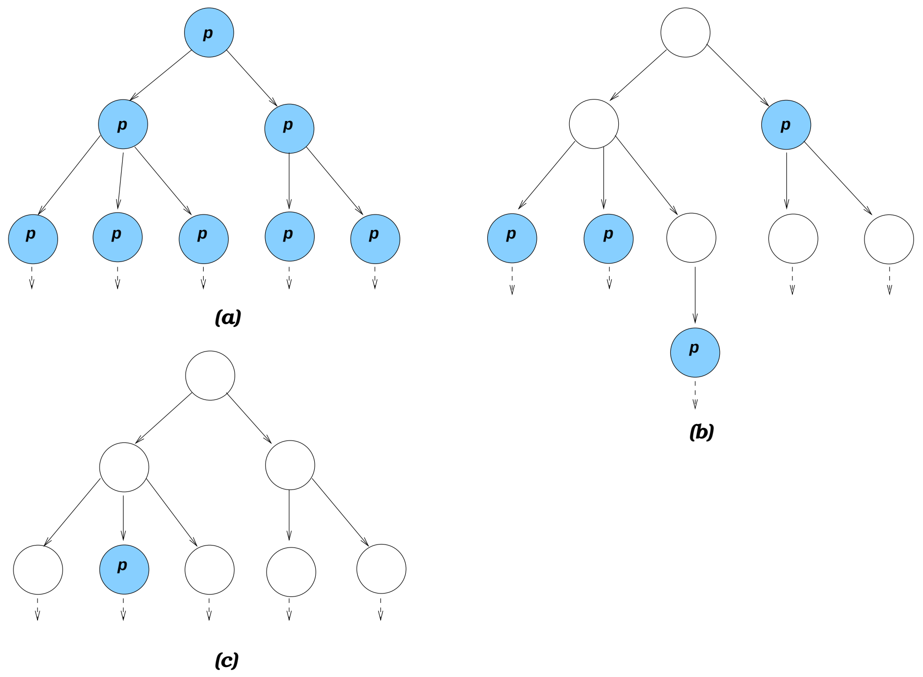


Figure 3.1: (a) AG(p): p is invariant. (b) AF(p): p is inevitable. (c) EF(p): p potentially holds.

Our experiments are largely based on symbolic and bounded model checking using LTL and CTL assertions. Therefore, we provide a basic description of the main concepts involved in model checking.

Chapter 4

Model Checking

4.1 Introduction

Model checking is an automatic technique, introduced by Clarke and Allen Emerson [34], for verifying finite-state reactive systems, such as sequential circuit designs and communication protocols. The finite-state systems are modelled as state-transition graphs and specifications are expressed in propositional logic.

The usual search procedure is to determine automatically if the specifications are satisfied by the state-transition graph. The model checker will either return with the answer true, indicating that the model satisfies the specification, or give a counterexample execution path that shows why the specification or formula is not satisfied.

Model checking was initially unable to deal with very large systems which had billions of states. This was called the *state explosion problem*. A first attempt to solve the state explosion problem is the technique called *symbolic model checking*.

4.2 Symbolic Model Checking

The introduction of symbolic model checking increased the capacity of model checking and made it a standard in the hardware industry [23]. The main idea behind it was to represent the transition relations in the state-transition graph using *Binary Decision Diagrams (BDDs)*. A binary decision diagram is a binary decision tree, optimised by merging isomorphic subtrees. Thus, a BDD is a directed acyclic graph (DAG) for representing boolean functions [32]. Boolean functions can only take values from $\{0, 1\}$. BDDs are generally a useful mechanism for proving specifications for the following reasons, identified in [49]:

- BDDs are a canonical representation for boolean functions so that equivalence tests are cheap.
- BDDs are especially good at handling the boolean quantification that is needed in the image computations.
- Sometimes even sets of large cardinality might have a compact BDD representation.
- Automata-theoretic methods can also be represented in symbolic form, using BDDs.

Let's consider the following Binary Decision Diagram pictured in [34]:

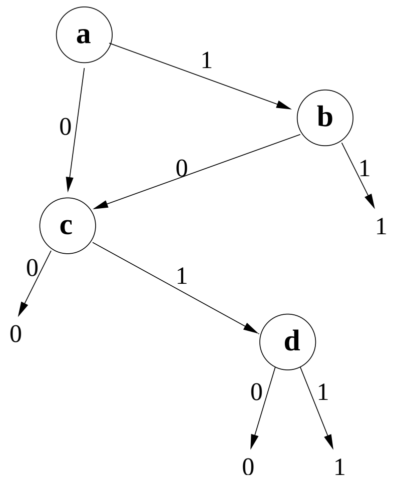


Figure 4.1: Binary Decision Diagram for $(a \wedge b) \vee (c \wedge d)$

If we would like to check the specification $(a \wedge b) \vee (c \wedge d)$, then given an assignment of boolean values to the variables a, b, c, d, it is possible to decide whether the assignment makes the formula true, by traversing the graph beginning at the root and branching at each node, based on the value assigned to the variable that labels the node. For example, the assignment $< a \leftarrow 1, b \leftarrow 0, c \leftarrow 1, d \leftarrow 1 >$, leads to a leaf node labelled 1, hence the formula is true for this assignment.

Therefore, by converting the transition relations to BDDs, a very concise representation of the finite state system can be obtained. It has been proved that symbolic model checkers that use BDD representations can sometimes process state spaces with more than 10^{120}

states [34]. State-of-the-art model checkers with high performances, used in our research are SAL and NuSMV.

4.2.1 SAL Symbolic Model Checker (SAL-SMC)

Given a SAL module of finite state space and a LTL formula, the SAL symbolic model checker decides whether the corresponding transition system satisfies the formula [48]. SAL-SMC uses BDDs to represent finite transition relations.

4.2.2 NuSMV

In NuSMV, the symbolic model checker also uses BDDs to represent the transition system. For each LTL specification, a tableau able to recognize the behaviours falsifying the property is built, and composed synchronously with the model [47].

Even though symbolic model checkers, such as SAL-SMC and NuSMV can sometimes process state spaces with more than 10^{120} states, the *state explosion problem* still exists. The bottleneck of symbolic model checking is the amount of memory required for storing and manipulating BDDs [26]. The boolean functions required to represent the set of states can grow exponentially, thus the size of the BDD representations may also explode during computation [48]. In these cases, symbolic model checking may fail to verify a small problem with 10^7 states, because there is no compact BDD representation for the underlying transition relation.

A new model checking mechanism called *bounded model checking* was first proposed by Biere et al. in 1999 [27]. This new technique was aimed at solving many cases that cannot be solved by BDD-based techniques.

4.3 Bounded Model Checking

The main idea in bounded model checking is to search for a counterexample in executions whose length is bounded by some integer k [26]. The bound k is increased until a bug is found, or some pre-computed *"completeness threshold"* is reached. The bounded model checking problem is efficiently reduced to a propositional satisfiability problem (SAT), and can therefore be solved by SAT methods rather than BDDs.

The state-of-the-art bounded model checkers, like SAL-BMC, SAL-INF-BMC and NuSMV

-bmc, use internal or external SAT solvers for verifying temporal logic properties. These bounded model checkers will try to generate counterexamples over computations of length k, for an invalid LTL or CTL property.

4.3.1 The bounded model checking mechanism

In general, a state-transition system $M = (X, I, T)$ is characterized by the following components [28]:

- its state space X.
- a set of initial states $I \subseteq X$.
- a transition relation $T \subseteq X \times X$.

Analysis tools such as SAL and NuSMV represent the set of initial states and the transition relation symbolically as two predicates $I(x)$ and $T(x, x^{'})$, where $x^{'}$ represents the value of x in the next state. The following two definitions stated in [28] are fundamental in defining the transition system:

Definition 5 *A state $x \in X$ is considered an initial state iff $I(x)$ holds.*

Definition 6 *A state $x^{'}$ is a successor of x by the transition relation T iff $T(x, x^{'})$ holds, i.e. $(x, x^{'}) \in T$.*

4.3.2 Safety properties

In its basic form, bounded model checking searches for counterexamples at depth $k \in N$ to a safety property P, represented symbolically via a predicate $P(x)$. Safety properties state that nothing bad will happen with certain states in the future. The model checkers use two different approaches in falsifying safety properties, as stated in [28].

The first approach is to find a finite sequence of states $x_0, x_1, \ldots, x_k$ that satisfies the formula: $\phi = I(x_0) \wedge T(x_0, x_1) \wedge \ldots \wedge T(x_{k-1}, x_k) \wedge \neg P(x_k)$
If such a sequence exists then we can conclude that the transition system M does not satisfy $\Box P$ (written GP in SAL and NuSMV), since x_k is reachable from the initial state x_0 and does not satisfy P. ϕ is unsatisfiable means that such a state x_k does not exist. However, the absence of counterexamples to $\Box P$ at depth k does not imply that no counterexamples

exist at a lesser depth.

Another approach is to find a finite sequence of states $x_0, x_1, \ldots, x_k$ that satisfies the formula: $\psi = I(x_0) \wedge T(x_0, x_1) \wedge \ldots \wedge T(x_{k-1}, x_k) \wedge \neg(P(x_0) \vee \ldots \vee P(x_k))$
If such a sequence exists, we can conclude as above that $\Box P$ is not satisfied. With this new formulation, and provided paths of length k exist, the absence of counterexamples at depth k implies the absence of counterexamples at lesser depths. If the system deadlocks, and has no trajectory of length k, then bounded model checking at depth k won't be able to find counterexamples for either formulation ϕ or ψ, but counterexamples may exist at lesser depths.

SAL-BMC and SAL-INF-BMC can be instructed to use *iterative deepening* (*-it* flag). Iterative deepening uses the ϕ formula to search for a counterexample of minimal length for $\Box P$. A sequence of SAT problems is fired and the satisfiability of ϕ is determined for $k = 0, k = 1$ and so forth until a counterexample is found or a maximal depth is reached. The default mode for both SAL-BMC and SAL-INF-BMC uses the formula ψ, and searches for its satisfiability for a user-specified depth k.

4.3.3 Invariance Properties

An invariant states that P *"always"* holds and is in the form $\Box P$. We have shown that invariants can be proved false by implicitly proving that the safety property P does not hold along a path $x_0, x_1, \ldots x_k$.

However, formulations like ϕ and ψ can be improved. It is sufficient to check that an invariant does not hold in at least one state x_i. Therefore, SAT solvers used with SAL-BMC, SAL-INF-BMC and NuSMV explicitly falsify invariants by checking the following formula, as stated in [48]:
$\gamma = I(x_0) \wedge T(x_0, x_1) \wedge \ldots \wedge T(x_{k-1}, x_k) \wedge (\neg P(x_0) \vee \ldots \vee \neg P(x_k))$

The formula γ is satisfiable iff there exists a path of length at most k from the initial state x_0, which violates the invariant $\Box P$. If bounded model checking cannot find a counterexample for an invariant $\Box P$, we cannot conclude in general that $\Box P$ is satisfied. Therefore, a powerful technique called induction or k-induction is used to fully prove invariants [23].

4.3.4 K-induction

The standard induction rule for proving an invariant $\Box P$ consists of proving that the following two formulas are valid [28]:

- Base Case: $I(x) \Rightarrow P(x)$
- Induction step: $P(x) \wedge T(x, x^{'}) \Rightarrow P(x^{'})$

This can be transformed into two satisfiability problems that can be solved using a SAT solver. SAL-BMC and SAL-INF-BMC can be instructed to prove an invariant $\Box P$ using k-induction, by using the -i flag. SAL will delegate the problem to its default SAT solver ICS or to an external SAT solver, who will either prove the LTL specification using standard induction, or will find a counterexample using the following k-induction rule [28]:

- Base Case : $I(x_0) \wedge T(x_0, x_1) \wedge \ldots \wedge T(x_{k-2}, x_{k-1}) \Rightarrow P(x_0) \wedge \ldots \wedge P(x_{k-1})$
- Induction Step: $P(x_0) \wedge T(x_0, x_1) \wedge \ldots \wedge T(x_{k-2}, x_{k-1}) \wedge P(x_{k-1}) \wedge T(x_{k-1}, x_k) \Rightarrow P(x_k)$

Chapter 5

Pathway Logic

5.1 Introduction

Research in biology and biomedicine has been revolutionized in recent years by the enormous growth of genomic sequence information and technological advances in the analysis of global gene expression [35]. There is a need for integrating that vast amount of experimental data and associated analyses into theoretical models of cellular processes for guiding hypothesis creation and testing.

Formal methods techniques have been used by various groups to develop executable models of biological systems at high levels of abstraction [50]. Typically, the techniques are based on a model of concurrent computation with associated formal languages for describing system behaviour and tools for simulation and analysis.

The *Petri net formalism* [45] has been developed to specify and analyse concurrent systems. Petri nets have a graphical representation that corresponds naturally to conventional representations of biochemical networks. They have been used to model metabolic pathways, simple genetic networks, and to map biochemical concepts such as stoichiometry, flux modes, and conservation relations to well-known Petri net theory concepts [50].

Other formal methods used to model biological systems are pi-calculus, a process algebra originally developed for describing concurrent computer processes, and statecharts, a visual notation for specifying reactive concurrent systems.

In this research paper, our experiments are largely based on Pathway Logic. Pathway Logic (PL) is an approach to modelling biological processes as Petri nets, based on formal methods and rewriting logic [5]. Representing biological knowledge using formal rules and

concepts allows data to be interpreted, combined, and queried in the context of biological knowledge.
State-of-the-art: Pathway Logic is currently being used for the modelling and analysis of signal transduction and metabolic networks in mammalian cells. PL models are represented using the Maude system [12], a system founded on rewriting logic.
Rewriting Logic [41] is a simple but powerful logical formalism based on two simple ideas:

- the states of a system are represented as elements of an algebraic data type
- the behaviour of a system is given by local transitions between states described by abstractions called rewrite rules

In Pathway Logic the algebraic data types are used to represent concepts from cell biology needed to model signalling processes, including intracellular proteins and biochemical modification of proteins [50]. Rewrite rules are used to model local processes within a cell or transmission of a signal across a cell membrane. The biological signalling network is represented as a collection of rewrite rules together with the algebraic declarations.

Using Pathway Logic, biological models can be described at very different levels of abstraction, dynamic pathways can be generated using search and model-checking techniques, and the rewriting logic representation of models meeting certain simple conditions can be transformed into Petri nets for analysis and visualization [51].

5.2 Pathway Logic Basics

As mentioned above, Pathway Logic models of biological processes are developed using the Maude system [12], a formal language and tool set based on rewriting logic.
PL models are structured in four layers, as mentioned in [51]:

(1) **Sorts and operations.** This layer defines the main sorts, subsort relations, and operations for representing cell states. The sorts of entities include *Chemical, Protein, DNA, Complex, and Enclosure* (cells and other compartments). These are all subsorts of the sort *Soup*, which is a multiset that represent liquid mixtures. Worth mentioning is the sort *Dish*, which encapsulates a *Soup* as a state to be observed. Post-translational protein modification is represented by terms of the form [P - mods],

where P is a protein and mods is a set of modifications. For example, the term [Cas - act] represents the activation of the protein Cas.

(2) **Components.** This layer specifies particular entities like proteins, chemicals, DNA, and introduces additional sorts for grouping proteins in families.

(3) **Rules.** This layer contains rewrite rules specifying individual steps such as metabolic reactions, activation or translocation.

(4) **Queries.** This layer specifies initial states and properties of interest.

5.3 The Pathway Logic Assistant

The Pathway Logic Assistant (PLA) [51] is a visualization and analysis tool that provides a user interface that supports visualization of and interaction with the biological models. The Interoperability Platform (IOP) [40] is the underlying infrastructure for the PLA.

The IOP project is aimed at developing an infrastructure for allowing formal tools like Maude and PVS to interact via simple, well defined, semantically meaningful communication interfaces. Currently, IOP facilitates the communication of IMaude, an interactive extension of Maude, with other tools, including other instances of itself, web resources, visualization tools, theorem provers such as PVS [15], as well as to read and write files, and execute shell commands.

Once a set of rules is represented in Maude, the biologists can use the Pathway Logic Assistant to explore the model structure and to ask questions, such as: "starting with a cell containing particular proteins and chemicals can a state be reached matching a particular pattern?". These kind of questions can be answered using execution, search, and model-checking in Maude, or by converting the model to a Petri net and using Petri net analysis tools. The Petri net is represented using an interactive network graph, in which the biological network, subnets, and generated pathways can be visualized. The interactive network graph has actions associated either with the graph as a whole or with particular nodes. For example, a particular signalling subnet can be found, for the activation of protein Fak. This can be acomplished by first choosing from the Selections menu, Fak-act as a goal. Goals, such as Fak-act, are coloured in green. Then the graph is asked to find a path leading to the goal state. A rule node can be selected, and has an action that will display the Maude code for the rule.

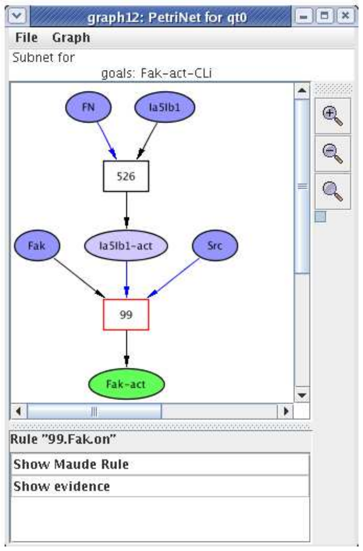
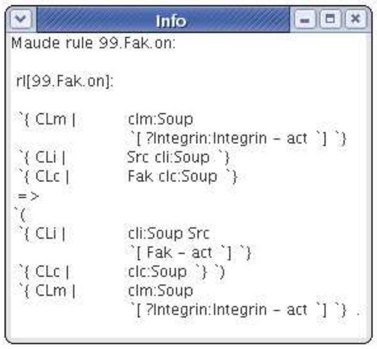


Figure 5.1: Pathway for the activation of protein Fak



```
Info
Maude rule 99.Fak.on:

rl[99.Fak.on]:

`{ CLm |        clm:Soup
                `[ ?Integrin:Integrin - act `] `}
`{ CLi |        Src cli:Soup `}
`{ CLc |        Fak clc:Soup `}
=>
`(
`{ CLi |        cli:Soup Src
                `[ Fak - act `] `}
`{ CLc |        clc:Soup `} `)
`{ CLm |        clm:Soup
                `[ ?Integrin:Integrin - act `] `} .
```

Figure 5.2: The result of invoking Show Maude Rule

Chapter 6

Modelling and Querying Cellular Signalling Networks

6.1 Introduction

There is a vast amount of experimental data in biology and biomedicine that needs to be analysed and interpreted. Thus, there is an urgent need to construct predictive models and to conduct in silico experiments using formal methods in general and rewriting logic based formalisms in particular [36].

A significant amount of work has been done investigating mammalian signalling processes and the molecular pathways by which cells detect, convert, and internally transmit information from their environment to intracellular targets such as the genome [35, 50, 36]. Our work is based on the computational models of mammalian signalling processes created by Maude [12] using rewriting logic principles, and exported as Petri nets representations.

The content of our work is described in the following way: this chapter describes the naming conventions used throughout the rest of this paper, gives an overview of signal transduction in mammalian cellular networks, explains the particular biological networks, and the types of queries used in our experiments. The next chapter contains the actual experiments, and provides an overview of the biological concepts used, such as Soup and Dish. Finally, the last chapter draws the appropriate conclusions, formulates rules of thumb and indicates future research directions.

6.2 Naming conventions

In this paper we will use the name conventions initiated and used by Merrill Knapp (SRI), the principal curator of PL models. The following naming conventions are used:

- **CLi** and **CLm** refer to the location of an element (protein , gene, chemical), which might be *inside* the cell or in its *membrane*.
- **NUc** refers to an element located in the cytosol of the nucleus.
- **act** indicates that a protein is activated.
- **on** is typically used as a gene modification to indicate that it is enabled for transcription (i.e. to produce the protein it encodes).

6.3 Signal transduction

Signal transduction represents any process by which a cell converts one kind of signal or stimulus into another. It usually involves a sequence of biochemical reactions inside the cell, which are carried out by enzymes and linked through second messengers (i.e. low weight diffusible molecules) [25].

In our work, we have used two models of signal transduction in mammalian cellular networks: the epidermal growth factor receptor (EGFR) signalling network, and the molecular interaction network of a macrophage.

6.3.1 EGFR network

The epidermal growth factor receptor (EGFR) signalling pathway is one of the most important pathways that regulate growth, survival, proliferation, and differentiation in mammalian cells. It is one of the best investigated signalling systems, both experimentally and computationally, and several computational models have been developed for dynamic analysis [43].

We are interested in finding signalling pathways that lead to the activation of various proteins, genes, biochemicals etc.

Activation of protein Rac1

Rac1 is a protein that becomes active when it has a GTP (guanosine triphosphate) bound to it. Rac1 becomes inactive when the GTP has been exchanged for GDP. This cycle acts as a molecular switch in signalling pathways by regulating functions of other proteins [44].

More specifically, by means of model checking, we are interested in finding pathways along which Rac1 is activated inside the cell. According to our naming conventions, we will consider all the pathways in which Rac1-GTP-CLi appears. An interesting aspect is the relation between the activation of Rac1 and other proteins, such as Vav2 and EgfR (i.e. the epidermal growth factor receptor). We intend to discover:

- Pathways leading to the activation of Rac1 which might use the protein Vav2 activated inside the cell (i.e. Vav2-act-CLi).
- Pathways in which the activation of EgfR in the cell membrane (i.e. EgfR-act-CLm) is not a necessary checkpoint for activating the protein Rac1.
- Pathways in which Rac1 can be activated without ever producing EgfR.

Production of chemicals PIP3 and IP3

We would also like to find out how the chemicals PIP3 (phosphatidylinositol) and IP3 (inositol) influence the activation of each other in the cell membrane. Hence, we intend to:

- Discover signalling pathways that lead to the production of PIP3 in the cell membrane (i.e. PIP3-CLm).
- Discover pathways in which PIP3-CLm is produced and later IP3-CLm is produced.
- Discover pathways that lead to the creation of IP3, in which PIP3 is not a necessary checkpoint.
- Find pathways in which IP3 is produced without ever producing PIP3.

6.3.2 The molecular interaction network of a macrophage

The biological interaction network used in our experiments is in fact a transliteration done by Merrill Knapp of one of Hiroaki Kitano's original cell designer examples [42].

The initial map of a macrophage is a comprehensive map of molecular interactions in a macrophage, developed with the purpose of indentifying intracellular molecular interactions and understanding complex dynamics and mechanisms of the cell. Macrophages are cells found in tissues that are responsible for phagocytosis (i.e. cell eating) of pathogens (i.e. infectious agents), dead cells and cellular debris. They are part of the innate immune system [24].

Our main interest is to find pathways that lead to the activation of the gene that codes for the protein Fos (proto-oncogene protein c-fos), named Fosgene. In particular, we are interested how the activation of the protein family Erk(Mitogen-activated protein kinase) inside the cell (i.e. Erk-act-CLi) influences the production of Fosgene in the cytosol of the nucleus (i.e. Fosgene-on-NUc). Thus we intend to find:

- Pathways leading to Fosgene-on-NUc which might use the activated protein Erk.
- Pathways in which the production of protein Erk is not a necessary checkpoint for turning Fosgene on (producing Fosgene-on-NUc).
- Pathways in which Fosgene can be activated without ever activating Erk.

Similar to the EGFR network study, we would also like to discover signalling pathways that lead to the activation of the chemical PIP3 in the cell membrane (i.e. PIP3-CLm).

6.4 Biological queries

The use of formal reasoning tools such as Maude, SAL and NuSMV to represent complex molecular networks is motivated by the promise of achieving biologically qualitative analyses of the dynamical behaviour of these networks. This is very important in the development of computational biology, for the following reasons, identified by Fages et al. in [29]:

- Qualitative analyses promise to provide logical and computational interpretation of the role of biologically relevant subparts of regulatory, signalling and metabolic networks. These networks are very complex mechanisms which are far from being understood on a global scale.

- Data on both the existence and dynamics of molecular interactions is rare and unreliable. Hence, dynamical models are too sensitive to the exact network structure and are not suited to analyze and predict the behaviour of molecular interactions.

In our experiments, we explore the use of automated methods for querying qualitative models of biomolecular networks, using the Temporal Logic framework. Our queries will be mainly Linear Temporal Logic queries, but we will make use of some Computation Tree Logic queries as well. The biological queries that biologists can consider about models of signal transduction in mammalian cellular networks are of different kinds. Below we enumerate a list of types of biological queries used in our work, and discuss their expression in LTL and CTL:

About simple reachability:

(1) Given an initial state *init*, is there a pathway for producing a species P? This query translates into the LTL formula $\mathbf{F}(\mathrm{P})$ and CTL formula $\mathbf{EF}(\mathrm{P})$. However, since we will be using bounded model checking, we will look for counterexamples of their duals: $\mathbf{G}(\neg(\mathrm{P}))$ and respectively $\mathbf{AG}(\neg(\mathrm{P}))$.

About pathways:

(2) Given an initial state *init*, can the cell reach a state P while passing by another state Q? A path that is a counterexample for the LTL formula $\mathbf{F}(\mathrm{P}) \Rightarrow \neg(\mathrm{Q})\ \mathbf{U}\ \mathrm{P}$ has the desired property. However, this is not a causal relation, since Q may not be actually causally used to produce P, it may just be coincidental. On the other hand, if the above formula is proved, and P is producable, then every path that produces P has the property that Q is not present before P (i.e. Q prevents the production of P).

(3) Given an initial state *init*, is it possible to produce a state P without using a state Q? A path that is a counterexample for the LTL formula $\mathbf{F}(\mathrm{P}) \Rightarrow \neg(\neg(\mathrm{Q})\ \mathbf{U}\ \mathrm{P})$ has the desired property. This is a causal relation, which means that if the formula is proved and P is producable, then the state Q is a necessary checkpoint for reaching state P. Biologists call Q in this case a "knockout" for P, because if Q is prevented then P is also prevented.

(4) Given an initial state *init*, is it possible to produce P without ever producing Q? A path that is a counterexample to the LTL formula $\mathbf{F}(\mathrm{P}) \Rightarrow \neg(\mathbf{G}(\neg(\mathrm{Q})))$ has the desired property.

Chapter 7

Computational results

7.1 Introduction

In our experiments, we compared the performances of various formal reasoning tools such as SAL, NuSMV, Maude and Lola, in modelling and querying the biological cellular signalling networks described in the previous chapter. We also aimed to find well argued rules of thumb that should be used as a guide when deciding which of the SAL or NuSMV tools would be best to use to answer a given query if integrated into the Pathway Logic Assistant.

Before presenting the experimental results, we find it necessary to describe in more detail the notions of Soup and Dish, defined in Maude.

7.2 The concepts of Soup and Dish

In eukaryotic cells (cells with a nucleus) proteins and other molecules exist in complex mixtures that are compartmentalized [35]. Such a compartmentalized mixture is called a *Soup*. In Maude, a soup is defined as a multiset of things. The multiset union operation models the presumed fluid or dynamic nature of some subcellular compartments, where the order in which molecules exist in the soups does not matter [36]. For example the term {*CM*| *cm:Soup PIP3 [Pdk1 - act]* {*cyto:Soup PKCe*}} represents a cell containing the chemical PIP3 and the activated protein [Pdk1 - act] in the cell membrane and the protein PKCe in the cytoplasm.

A particular sort of top-level soup called *Dish* serves as a container for carrying out in silico experiments. The dishes used in these experiments are translated into boolean models, which range in size and complexity from small to medium and finally large dishes. The models were developed at the SRI. We will use two dishes for analyzing the EGFR network and one dish for analyzing the macrophage network.

The first dish, used in the EGFR network, is called *"ThreeWaysToActRac"*. It is a small sized dish. Its corresponding boolean model has 39 transitions and 69 species. *"ThreeWaysToActRac"* was intended to show that the protein Rac1 could be activated by an EGF stimulus by three different routes.

The second dish, also used in the EGFR network, is called *"EgfDemo"*. It is a medium sized dish. Its corresponding boolean model has 63 transitions and 103 species. We have used this dish for finding signalling pathways that lead to the production of the chemicals PIP3 and IP3, as explained in the previous chapter.

The third dish, used in the molecular interaction network of a macrophage, is called for historical reasons *"Kitano Macrophage"*. It is a large sized dish. Its corresponding boolean model has 342 transitions and 543 species. We have used this dish for finding signalling pathways that lead to the production of the chemical PIP3 and the activation of the DNA Fosgene, using the activated protein family Erk.

7.3 Experiments

Below are the experiments on which we have based our conclusions. All the experiments were done on a machine running Fedora Core 3, with an Athlon 1.9 Gh processor and 512 MB core memory.

We have used two Perl scripts named *salParser4.pl* and *smvParser.pl*. Both scripts take as input a file containing a Petri net representation of the biological network, generated by Maude.

The scripts produce SAL and respectively NuSMV specifications that model the interaction rules within the given Petri net, and allow biologists to address queries, using LTL and CTL formulas. This is done by translating a specification produced from curated Maude models. One feature of salParser4.pl is that it generates a deadlock transition, one that is only enabled if no other transition is enabled, and that does not change the state. This is necessary in order to preserve the Maude semantics, which treats finite computations as infinite computations that end with an infinity of identity transitions. NuSMV has a built-in mechanism for detecting deadlocks.

We have used different SAT solvers in conjunction with the bounded model checker within SAL (sal-bmc). Thus, we were able to speed up the computation process of sal-bmc by using SAT solvers like Siege, BerkMin and zChaff. A minor drawback is that currently only zChaff is available on both Macintosh and Linux platforms.

Type	Query	**SAL-SMC**		**SAL-BMC**		**NuSMV**		**Lola**		**Maude**	
		Time secs	Path length	Time secs	Path length	Time secs	Path length	Time secs	Path length	Time secs	Path length
1	**G**(¬ (Rac1-GTP-CLi))	11.94	6	2.52	8	<1	5	<1	5	<1	5
2	**F**(Rac1-GTP-CLi)⇒ ¬(Vav2-act-CLi) **U** Rac1-GTP-CLi	5.38	7	1.79	7	1.1	6	–	–	<1	5
3	**F**(Rac1-GTP-CLi)⇒ ¬(¬(EgfR-act-CLm) **U** Rac1-GTP-CLi)	5.8	9	1.22	9	1.12	6	–	–	<1	10
4	**F**(Rac1-GTP-CLi)⇒ ¬**G**(¬(EgfR-act-CLm))	5.85	–	7.88	–	<1	8	–	–	<1	8

Table 7.1: Evaluation of LTL queries in the small dish "ThreeWaysToActRac"

Type	Query	**SAL-SMC**		**SAL-BMC**		**NuSMV**		**Lola**		**Maude**	
		Time secs	Path length	Time secs	Path length	Time secs	Path length	Time secs	Path length	Time secs	Path length
1	**G**(¬(PIP3-CLm))	16.4	4	1.5	5	<1	4	1.5	4	<1	46
1	**AG**(¬(PIP3-CLm))	16.4	4	1.49	5	–	–	–	–	–	–
2	**F**(IP3-CLm)⇒ ¬(PIP3-CLm) **U** IP3-CLm	17.75	8	1.84	10	1.5	7	–	–	<1	46
3	**F**(IP3-CLm)⇒ ¬(¬(PIP3-CLm) **U** IP3-CLm)	17.52	5	1.78	10	1.2	4	–	–	<1	37
4	**F**(IP3-CLm)⇒ ¬(**G**(¬(PIP3-CLm)))	74.44	39	239.88	39	1.3	4	–	–	<1	37

Table 7.2: Evaluation of LTL and CTL queries in the medium dish "EgfDemo"

Type	Query	**SAL-SMC**		**SAL-BMC**		**NuSMV**		**Maude**	
		Time secs	Path length	Time secs	Path length	Time secs	Path length	Time secs	Path length
1	**G**(¬(PIP3-CLm))	–	–	48.93	10	2.5	3	<1	>100
1	**G**(¬(Fosgene-on-NUc))	–	–	50.86	10	3.5	3	<1	>100
1	**AG**(¬(Fosgene-on-NUc))	–	–	55.22	6	–	–	–	–
2	**F**(Fosgene-on-NUc)⇒ ¬(Erk-act-CLi) **U** Fosgene-on-NUc	–	–	194.66	24	9.4	7	2.8	>100
3	**F**(Fosgene-on-NUc)⇒ ¬(¬(Erk-act-CLi) **U** Fosgene-on-NUc)	–	–	52.12	9	4.3	3	2.2	>100
4	**F**(Fosgene-on-NUc)⇒ ¬(**G**(¬(Erk-act-CLi)))	–	–	66.73	10	5.4	3	–	–

Table 7.3: Evaluation of LTL and CTL queries in the large dish “Kitano Macrophage”

The above experiments represent the execution times and lengths of the counterexamples generated by formal reasoning tools such as SAL, NuSMV, Maude and Lola [1], as a result of applying model checking on different kinds of dishes. These timings are sometimes slower than what is usually expected in the program verification community. The main reason for this is the overall structure of transition relations modelling biomolecular networks. Such transition systems are highly non-deterministic due to the *“soup”* aspect of molecular interactions.

The timings obtained with the NuSMV model checker show the efficiency of this tool, when compared with the others. One reason for this might be related to the fact that NuSMV seems to find the shortest signalling pathway or counterexample, without entering any unnecessary loops.

The results obtained with the SAL model checker show that sal-bmc takes full advantage of external SAT solvers like Siege, BerkMin or zChaff, having execution times comparable with NuSMV. However, the execution paths generated by SAL are usually longer than NuSMV’s. Sometimes (see the last row from Table 7.2) SAL enters unnecessary deadlock loops.

Lola proves to be a reliable tool when it is dealing with simple reachability queries (i.e. type 1). However, the results of the queries with types 2, 3 and 4 were obtained by inspecting the execution paths, using the Pathway Logic Assistant. Thus, these results are rather uninformative and are not included in the tables.

The timings obtained with the Maude model checker show that this tool is the fastest one when it terminates, but does not always have the best answer. Maude adopts a depth-first algorithm when generating counterexamples and not a breadth-first algorithm like NuSMV and SAL. Therefore, Maude usually generates huge counterexamples with tens of redundant transitions.

Chapter 8

Conclusions and Suggestions for Further Work

We have presented in this thesis how formal reasoning tools can be used to model complex non-deterministic mammalian biomolecular signalling networks. Using Perl scripts, we were able to generate SAL and NuSMV specifications that model the cellular networks. We did this by translating a Petri net specification produced from curated Maude models.

Four different representations of Petri nets were tried when creating SAL specifications. The first one identifies each SAL module with a transition rule and composes all the modules asynchronously. The second one uses a single SAL module in which the transition rules are simply enumerated using SAL's choice operator []. The third one also uses a single SAL module, but stores the components of the system in an array of booleans. The fourth and final version used throughout our experiments, generates a deadlock transition, one that is only enabled if no other transition is enabled. The reasons for using the fourth version were explained in the previous chapter.

We have used the Temporal Logic framework to query these systems. Interesting results were obtained, which should be used as a guide when assessing the suitability of SAL and NuSMV for the integration in the next release of the Pathway Logic Assistant.

We found it appropriate to state a few rules of thumb that characterize the main features of each formal reasoning tool used in our work.

General rules of thumb:

(1) For simple reachability queries of type $\mathbf{G}(\neg(\mathrm{P}))$, applied on small, medium or large dishes, it is advisable to try Lola first, then Maude.

(2) For queries of type $\mathbf{F}(\mathrm{P}) \Rightarrow \neg(\mathrm{Q})\ \mathbf{U}\ \mathrm{P}$, applied on small, medium or large dishes, it is advisable to use NuSMV first and then try SAL.

(3) For queries of type $\mathbf{F}(\mathrm{P}) \Rightarrow \neg(\neg(\mathrm{Q})\ \mathbf{U}\ \mathrm{P})$, which we call *necessity queries*, applied on all kinds of dishes, it is advisable to use NuSMV first and then try SAL.

(4) For queries of type $\mathbf{F}(\mathrm{P}) \Rightarrow \neg(\mathbf{G}(\neg(\mathrm{Q})))$, it is advisable to first use NuSMV for all kinds of dishes. For small dishes, one can also use Maude. For medium dishes, after trying NuSMV, one can try Maude. Finally, for large dishes, the second choice should be SAL.

We also implemented and integrated two classes of SAL actors in the Interoperability Platform (IOP). The intended outcome is to make communication possible between SAL and other formal reasoning tools, such as Maude and PVS, which are already part of IOP. We explain the difficulties and reasons for implementing two classes of actors, by identifying two different categories of SAL tools: the first category contains programs that terminate after executing a request, and the second category contains systems programmed as a “read-eval-print” loop.

There are many more improvements to be made in the implementation of the SAL and NuSMV parsers. We found that using the current specifications, SAL and NuSMV have weak performances, when dealing with CTL queries. SAL is able to answer only to simple reachability queries expressed in CTL, while NuSMV doesn’t respond to any type of CTL query. The main improvement would be to find more efficient SAL and NuSMV specifications, which are able to handle CTL queries. In order to test these changes on different dishes, the Perl parsers should be updated appropriately.
Another future work item would be to prove the correctness of the mapping from Petri nets to SAL and NuSMV specifications.

Bibliography

[1] The Lola Homepage — `http://www.informatik.hu-berlin.de/%7Ekschmidt/lola.html`.

[2] Introduction in Logic — `http://www.pst.informatik.uni-muenchen.de/lehre/SS02/tl/unterlagen/tl1-3-remote.pdf`.

[3] Introduction to Formal Methods — `http://www.dacs.dtic.mil/techs/2fmethods/intro.shtml`.

[4] Logic in Software Engineering — `http://www.cas.mcmaster.ca/~lawford/2F03/`.

[5] SRI Computer Science Laboratory — `http://www.csl.sri.com/`.

[6] The BerkMin Homepage — `http://eigold.tripod.com/BerkMin.html`.

[7] The CVC Homepage — `http://verify.stanford.edu/CVC/more.html`.

[8] The CVC-Lite Homepage — `http://verify.stanford.edu/CVCL/`.

[9] The ICS Homepage — `http://ics.csl.sri.com/`.

[10] The IOP Homepage — `http://mcs.une.edu.au/~iop/`.

[11] The IOP Manual — `http://mcs.une.edu.au/~iop/Data/Manual/manual.pdf`.

[12] The Maude Homepage — `http://maude.cs.uiuc.edu/`.

[13] The NuSMV Homepage — `http://nusmv.irst.itc.it/`.

[14] The Pathway Logic Assistant Homepage — `http://www.csl.sri.com/users/clt/PLweb/pl.html`.

[15] The PVS Homepage — `http://pvs.csl.sri.com/`.

[16] The SAL Homepage — `http://sal.csl.sri.com/`.

[17] The Siege Homepage — `http://www.cs.sfu.ca/~loryan/personal/`.

[18] The SVC Homepage — `http://verify.stanford.edu/SVC/`.

[19] The UCLID Homepage — `http://www.cs.cmu.edu/~uclid/`.

[20] The zChaff Homepage — `http://www.princeton.edu/~chaff/zchaff.html`.

[21] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[22] Gul Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. A Foundation for Actor Computation. *Journal of Functional Programming*, 7(1):1–72, 1997.

[23] Roy Armoni, Limor Fix, Ranan Fraer, Scott Huddleston, Nir Piterman, and Moshe Y. Vardi. SAT-based Induction for Temporal Safety Properties. *Electr. Notes Theor. Comput. Sci.*, 119(2):3–16, 2005.

[24] Irina Baranova, Tatyana Vishnyakova, Alexander Bocharov, Zhigang Chen, Alan T. Remaley, John Stonik, Thomas L. Eggerman, and Amy P. Patterson. *Infection and Immunity*, volume 70. June 2002.

[25] Jeremy M. Berg, John L. Tymoczko, Lubert Stryer, and Neil D. Clarke. *Biochemistry.* 2002. `http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=stryer`.

[26] A. Biere, A. Cimatti, Edmund Clarke, Ofer Strichman, and Y. Zhu. Bounded Model Checking . Number 60 in Advances In Computers, 2003. (to appear).

[27] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, London, UK, 1999. Springer-Verlag.

[28] Bruno Dutertre and Maria Sorea. Timed Systems in SAL. Technical Report SRI-SDL-04-03, Computer Science Laboratory, SRI International, Menlo Park, CA, October 2004.

[29] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos, Francois Fages, and Vincent Schachter. Modeling and Querying Biomolecular Interaction Networks. *Theor. Comput. Sci.*, 325(1):25–44, 2004.

[30] A. Cimatti, E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Integrating BDD-based and SAT-based Symbolic Model Checking. In *Proc. "Frontiers of Combining Systems, FROCOS'02"*, volume 2309 of *LNAI*, Santa Margherita, Italy, January 2002. Springer.

[31] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[32] Leonardo de Moura. Stanford Little Engines Course. Lecture 4, 2003.

[33] Leonardo de Moura, Sam Owre, Harald Ruess, John Rushby, N. Shankar, Maria Sorea, and Ashish Tiwari. SAL 2. In Rajeev Alur and Doron Peled, editors, *Computer-Aided Verification, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.

[34] D. Long Edmund M. Clarke, O. Grumberg. Model Checking. In *FSTTCS*, pages 54–56, 1997.

[35] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, José Meseguer, and M. Kemal Sönmez. Pathway Logic: Symbolic Analysis of Biological Signaling. In *Pacific Symposium on Biocomputing*, pages 400–412, 2002.

[36] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln, and Carolyn Talcott. Pathway Logic: Executable Models of Biological Networks. In *Fourth International Workshop on Rewriting Logic and Its Applications (WRLA'2002), Pisa, Italy, September 19 — 21, 2002*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002. `http://www.elsevier.nl/locate/entcs/volume71.html`.

[37] Formal Methods Program. Formal Methods Roadmap: PVS, ICS, and SAL. Technical Report SRI-CSL-03-05, Computer Science Laboratory, SRI International, Menlo Park, CA, October 2003.

[38] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 163–173, January 1980.

[39] Carl Hewitt, Peter Bishop, and Richard Steiger. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *IJCAI*, pages 235–245, 1973.

[40] Ian A. Mason and Carolyn L. Talcott. IOP: The InterOperability Platform & IMaude: An Interactive Extension of Maude. *Electr. Notes Theor. Comput. Sci.*, 117:315–333, 2005.

[41] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. In *Selected papers of the Second Workshop on Concurrency and compositionality*, pages 73–155, Essex, UK, 1992. Elsevier Science Publishers Ltd.

[42] Kanae Oda, Tomomi Kimura, Yukiko Matsuoka, Akira Funahashi, Masaaki Muramatsu, and Hiroaki Kitano. Molecular Interaction Map of a Macrophage. Technical report, The Systems Biology Institute, Tokyo, Japan, August 2004.

[43] Kanae Oda, Yukiko Matsuoka, Akira Funahashi, and Hiroaki Kitano. A comprehensive pathway map of epidermal growth factor receptor signalling. Technical report, The Systems Biology Institute, Tokyo, Japan, May 2005.

[44] Crespo P, Schuebel KE, Ostrom AA, Gutkind JS, and Bustelo XR. Phosphotyrosine-dependent activation of Rac-1 GDP/GTP exchange by the vav proto-oncogene product. *Nature*, pages 169–172, 1997.

[45] J. L. Peterson. *Petri Nets: Properties, analysis and applications.* Prentice-Hall, 1981.

[46] A. Pnueli. The temporal logic of programs. *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[47] Roberto Cavada and Alessandro Cimatti and Emanuele Olivetti and Gavin Keighren and Marco Pistore and Marco Roveri. NuSMV 2.2 User Manual. Technical report, IRST, Trento, Italy, 2005. `http://nusmv.irst.itc.it/`.

[48] Harald Ruess and Leonardo de Moura. From Simulation to Verification (and Back). In S. Chick, P.J. Sanche, D. Ferrin, and D.J. Morrice, editors, *WSC'03—Winter Simulation Conference*, New Orleans, LA, December 2003.

[49] Natarajan Shankar. Symbolic Analysis of Transition Systems. In *ASM '00: Proceedings of the International Workshop on Abstract State Machines, Theory and Applications*, pages 287–302, London, UK, 2000. Springer-Verlag.

[50] C. Talcott, S. Eker, M. Knapp, P. Lincoln, and K. Laderoute. Pathway Logic Modeling of Protein Functional Domains in Signal Transduction. In *Proceedings of the Pacific Symposium on Biocomputing*, January 2004.

[51] Carolyn Talcott and David L. Dill. The Pathway Logic Assistant. In Gordin Plotkin, editor, *Third International Workshop on Computational Methods in Systems Biology*, pages 228–239, 2005.

[52] Tom Laureys. Event-Based Semantics to Linear Temporal Logic. Master's thesis, School of Cognitive Science, University of Edinburgh, Edinburgh, UK, 1999. `http://www.ltg.ed.ac.uk/prosper/papers/laureys-1999-ebs/laureys-1999-ebs.pdf`.

Appendix A

SAL , NuSMV, Maude and Lola experiments

The following experiments show the execution times and counterexamples generated by SAL, NuSMV, Maude and Lola. The types of dishes and queries used are contained in the caption of each table.

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		11.94	1.EgfR.on 478.Vav2.on 40.Cdc42.by.Vav2 67.Pi3k.by.Cdc42 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
SAL-BMC	Siege	0.87	1.EgfR.on 4.Cbl.by.EgfR 870.Crk.by.Cbl 761.Grb2.to.EGFR 853.Pi3k.by.Cbl 643.PIP3.from.PIP2.by.Pi3k 478.Vav2.on 171.C3g.on 890.Rac1.on-1
	BerkMin	0.81	526.Ia5Ib1.on 1.EgfR.on 4.Cbl.by.EgfR 761.Grb2.to.EGFR 478.Vav2.on 11.Sos1.on 800.Sos1.Off 853.Pi3k.by.Cbl 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
	zChaff	1.22	526.Ia5Ib1.on 99.Fak.on 1.EgfR.on 478.Vav2.on 761.Grb2.to.EGFR 40.Cdc42.by.Vav2 67.Pi3k.by.Cdc42 4.Cbl.by.EgfR 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
	ICS	2.52	1.EgfR.on 526.Ia5Ib1.on 761.Grb2.to.EGFR 4.Cbl.by.EgfR 853.Pi3k.by.Cbl 643.PIP3.from.PIP2.by.Pi3k 478.Vav2.on 890.Rac1.on-1
NuSMV	SIM	<1	1.EgfR.on 478.Vav2.on 40.Cdc42.by.Vav2 67.Pi3k.by.Cdc42643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
Maude		<1	1.EgfR.on 67.Pi3k.by.Cdc42 478.Vav2.on 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
Lola		<1	1.EgfR.on 67.Pi3k.by.Cdc42 478.Vav2.on 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1

Table A.1: “ThreeWaysToActRac” dish – $\mathbf{G}(\neg$ (Rac1-GTP-CLi))

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		5.38	1.EgfR.on 478.Vav2.on 4.Cbl.by.EgfR 853.Pi3k.by.Cbl 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1 870.Crk.by.Cbl
SAL-BMC	Siege	0.92	1.EgfR.on 478.Vav2.on 4.Cbl.by.EgfR 526.Ia5Ib1.on 761.Grb2.to.EGFR 853.Pi3k.by.Cbl 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1 353.Gab1.on
	BerkMin	1.03	1.EgfR.on 478.Vav2.on 40.Cdc42.by.Vav2 67.Pi3k.by.Cdc42 761.Grb2.to.EGFR 643.PIP3.from.PIP2.by.Pi3k 108.Pdk1.by.PIP3 353.Gab1.on 890.Rac1.on-1 565.Pak.on
	zChaff	1.27	1.EgfR.on 4.Cbl.by.EgfR 761.Grb2.to.EGFR 870.Crk.by.Cbl 478.Vav2.on 40.Cdc42.by.Vav2 853.Pi3k.by.Cbl 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1 11.Sos1.on
	ICS	1.79	1.EgfR.on 478.Vav2.on 40.Cdc42.by.Vav2 67.Pi3k.by.Cdc42 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1 566.Pak.on
NuSMV	SIM	1.1	1.EgfR.on 478.Vav2.on 40.Cdc42.by.Vav2 67.Pi3k.by.Cdc42 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
Maude		<1	1.EgfR.on 67.Pi3k.by.Cdc42 478.Vav2.on 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
Lola		<1	1.EgfR.on 67.Pi3k.by.Cdc42 478.Vav2.on 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1

Table A.2: "ThreeWaysToActRac" dish – **F**(Rac1-GTP-CLi)$\Rightarrow$ $\neg$(Vav2-act-CLi) **U** Rac1-GTP-CLi

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		5.8	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 365.Crk.by.Pax 440.Cas.on 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3 343.Sos1.on
SAL-BMC	Siege	0.97	526.Ia5Ib1.on 99.Fak.on 533.Pi3k.by.Fak 434.Pax.phos 365.Crk.by.Pax 440.Cas.on 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3 343.Sos1.on
	BerkMin	1.01	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 365.Crk.by.Pax 563.Dock.by.Crk 38.Elmo.by.Dock 533.Pi3k.by.Fak 440.Cas.on 256.Rac1.on-3 566.Pak.on
	zChaff	1.22	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 365.Crk.by.Pax 563.Dock.by.Crk 440.Cas.on 38.Elmo.by.Dock 256.Rac1.on-3 1.EgfR.on
	ICS	2.11	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 440.Cas.on 365.Crk.by.Pax 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3 343.Sos1.on
NuSMV	SIM	1.12	1.EgfR.on 478.Vav2.on 40.Cdc42.by.Vav2 67.Pi3k.by.Cdc42 643.PIP3.from.PIP2.by.Pi3k 890.Rac1.on-1
Maude		<1	1.EgfR.on 526.Ia5Ib1.on 99.Fak.on 4.Cbl.by.EgfR 434.Pax.phos 870.Crk.by.Cbl 440.Cas.on 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3
Lola		<1	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 440.Cas.on 365.Crk.by.Pax 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3

Table A.3: "ThreeWaysToActRac" dish – $\mathbf{F}$(Rac1-GTP-CLi)$\Rightarrow\neg(\neg$(EgfR-act-CLm) $\mathbf{U}$ Rac1-GTP-CLi)

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		5.85	proved
SAL-BMC	Siege	7.88	no counterexample between depths: [0, 60]
	BerkMin	15.34	no counterexample between depths: [0, 70]
	zChaff	23.81	no counterexample between depths: [0, 80]
	ICS	78.67	no counterexample between depths: [0, 60]
NuSMV	SIM	<1	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 365.Crk.by.Pax 563.Dock.by.Crk 440.Cas.on 38.Elmo.by.Dock 256.Rac1.on-3
Maude		<1	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 440.Cas.on 365.Crk.by.Pax 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3
Lola		<1	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 440.Cas.on 512.Crk.by.Cas 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3

Table A.4: "ThreeWaysToActRac" dish – $\mathbf{F}$(Rac1-GTP-CLi)$\Rightarrow\neg(\mathbf{G}(\neg$(EgfR-act-CLm)))

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		16.4	526.Ia5Ib1.on 99.Fak.on 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K
SAL-BMC	Siege	1.67	1.EgfR.on 16.Shc.by.EgfR 172.Shp1.by.EgfR 761.Grb2.by.EgfR 353.Gab1.on 429.PI3K.by.Gab1 11.Sos1.by.Grb2 643.PI345P3.from.PI45P2.by.PI3K
	BerkMin	1.5	526.Ia5Ib1.on 99.Fak.on 713.Ilk.by.Integrin 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K
	zChaff	2.28	584.Eif4e.xby.4Ebp1 1.EgfR.on 526.Ia5Ib1.on 99.Fak.on 761.Grb2.by.EgfR 4.Cbl.by.EgfR 12.Plcg1.by.Fak 533.PI3K.by.Fak 54.Ca++.by.Integrin 643.PI345P3.from.PI45P2.by.PI3K
	ICS	29.55	1.EgfR.on 761.Grb2.by.EgfR 353.Gab1.on 429.PI3K.by.Gab1 643.PI345P3.from.PI45P2.by.PI3K
NuSMV	SIM	<1	526.Ia5Ib1.on 99.Fak.on 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K
Maude		<1	'1.EgfR.on '526.Ia5Ib1.on '54.Ca++.by.Integrin '4.Cbl.by.EgfR '870.Crk.by.Cbl '563.Dock.by.Crk '38.Elmo.by.Dock '99.Fak.on '761.Grb2.by.EgfR '353.Gab1.on '713.Ilk.by.Integrin '434.Pax.phos '440.Cas.on '853.PI3K.by.Cbl '322B.Eif4g.by.PI3K '643.PI345P3.from.PI45P2.by.PI3K '108.Pdk1.by.PIP3 '109.Akt.on '122.Gsk3.xby.Akt '107.PI45P2.from.PI345P3.by.Pten '567.PKA.xby.Integrin '643.PI345P3.from.PI45P2.by.PI3K '111.aPkc.on '107.PI45P2.from.PI345P3.by.Pten '12.Plcg1.by.Fak '84.IP3.by.Plc '314.cPkc.on '758.nPkc.on-1 '16.Shc.by.EgfR '172.Shp1.by.EgfR '167.Shp2.by.Gab1 '343.Sos1.on '121.Akt.to.nuc '561.Cav1.by.Integrin '256.Rac1.on-3 '473.Mekk1.by.Rac1 '148.Mek.by.Mekk1 '472.Mekk4.by.Rac1 '372.Mkk4.by.Mekk1 '566.Pak.on '711.Mkk3/6.by.Pak '61.P38.by.Mkk3 '448.P38.to.nuc '64.Elk1.by.P38 '263.Msk.by.P38 '442.Creb.by.Msk deadlock
Lola		1.5	1.EgfR.on 4.Cbl.by.EgfR 853.Pi3k.by.Cbl 643.PIP3.from.PIP2.by.Pi3k

Table A.5: “EgfDemo” dish – $\mathbf{G}(\neg(\text{PIP3-CLm}))$

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		16.4	526.Ia5Ib1.on 99.Fak.on 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K
SAL-BMC	Siege	1.66	584.Eif4e.xby.4Ebp1 612.Eps8.by.Actin 498.E3b1.by.Eps8 1.EgfR.on 761.Grb2.by.EgfR 353.Gab1.on 429.PI3K.by.Gab1 643.PI345P3.from.PI45P2.by.PI3K
	BerkMin	1.49	526.Ia5Ib1.on 99.Fak.on 713.Ilk.by.Integrin 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K
	zChaff	2.24	584.Eif4e.xby.4Ebp1 1.EgfR.on 526.Ia5Ib1.on 99.Fak.on 761.Grb2.by.EgfR 4.Cbl.by.EgfR 12.Plcg1.by.Fak 533.PI3K.by.Fak 54.Ca++.by.Integrin 643.PI345P3.from.PI45P2.by.PI3K
	ICS	29.55	1.EgfR.on 761.Grb2.by.EgfR 353.Gab1.on 429.PI3K.by.Gab1 643.PI345P3.from.PI45P2.by.PI3K
NuSMV	SIM	–	failed to timely terminate
Maude		–	–
Lola		–	–

Table A.6: “EgfDemo” dish – **AG**(¬(PIP3-CLm))

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		17.75	526.Ia5Ib1.on 99.Fak.on 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K 107.PI45P2.from.PI345P3.by.Pten 12.Plcg1.by.Fak 84.IP3.by.Plc 322B.Eif4g.by.PI3K
SAL-BMC	Siege	1.84	526.Ia5Ib1.on 54.Ca++.by.Integrin 99.Fak.on 1.EgfR.on 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K 12.Plcg1.by.Fak 107.PI45P2.from.PI345P3.by.Pten 84.IP3.by.Plc 567.PKA.xby.Integrin
	BerkMin	1.83	584.Eif4e.xby.4Ebp1 612.Eps8.by.Actin 526.Ia5Ib1.on 99.Fak.on 12.Plcg1.by.Fak 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K 107.PI45P2.from.PI345P3.by.Pten 84.IP3.by.Plc 561.Cav1.by.Integrin
	zChaff	2.5	526.Ia5Ib1.on 99.Fak.on 584.Eif4e.xby.4Ebp1 12.Plcg1.by.Fak 533.PI3K.by.Fak 612.Eps8.by.Actin 643.PI345P3.from.PI45P2.by.PI3K 107.PI45P2.from.PI345P3.by.Pten 84.IP3.by.Plc 1.EgfR.on
	ICS	4.89	526.Ia5Ib1.on 713.Ilk.by.Integrin 99.Fak.on 533.PI3K.by.Fak 643.PI345P3.from.PI45P2.by.PI3K 54.Ca++.by.Integrin 12.Plcg1.by.Fak 107.PI45P2.from.PI345P3.by.Pten 84.IP3.by.Plc 314.cPkc.on
NuSMV	SIM	1.5	526.Ia5Ib1.on 99.Fak.on 533.PI3K.by.Fak 12.Plcg1.by.Fak 643.PI345P3.from.PI45P2.by.PI3K 107.PI45P2.from.PI345P3.by.Pten 84.IP3.by.Plc
Maude		<1	’1.EgfR.on ’526.Ia5Ib1.on ’54.Ca++.by.Integrin ’4.Cbl.by.EgfR ’870.Crk.by.Cbl ’563.Dock.by.Crk ’38.Elmo.by.Dock ’99.Fak.on ’761.Grb2.by.EgfR ’353.Gab1.on ’713.Ilk.by.Integrin ’434.Pax.phos ’440.Cas.on ’853.PI3K.by.Cbl ’322B.Eif4g.by.PI3K ’643.PI345P3.from.PI45P2.by.PI3K ’108.Pdk1.by.PIP3 ’109.Akt.on ’122.Gsk3.xby.Akt ’107.PI45P2.from.PI345P3.by.Pten ’567.PKA.xby.Integrin ’643.PI345P3.from.PI45P2.by.PI3K ’111.aPkc.on ’107.PI45P2.from.PI345P3.by.Pten ’12.Plcg1.by.Fak ’84.IP3.by.Plc ’314.cPkc.on ’758.nPkc.on-1 ’16.Shc.by.EgfR ’172.Shp1.by.EgfR ’167.Shp2.by.Gab1 ’343.Sos1.on ’121.Akt.to.nuc ’561.Cav1.by.Integrin ’256.Rac1.on-3 ’473.Mekk1.by.Rac1 ’148.Mek.by.Mekk1 ’472.Mekk4.by.Rac1 ’372.Mkk4.by.Mekk1 ’566.Pak.on ’711.Mkk3/6.by.Pak ’61.P38.by.Mkk3 ’448.P38.to.nuc ’64.Elk1.by.P38 ’263.Msk.by.P38 ’442.Creb.by.Msk deadlock
Lola		–	–

Table A.7: “EgfDemo” dish – **F**(IP3-CLm)⇒ ¬(PIP3-CLm) **U** IP3-CLm

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		17.52	526.Ia5Ib1.on 99.Fak.on 12.Plcg1.by.Fak 84.IP3.by.Plc 533.PI3K.by.Fak
SAL-BMC	Siege	1.78	1.EgfR.on 172.Shp1.by.EgfR 612.Eps8.by.Actin 16.Shc.by.EgfR 526.Ia5Ib1.on 4.Cbl.by.EgfR 99.Fak.on 12.Plcg1.by.Fak 84.IP3.by.Plc 567.PKA.xby.Integrin
	BerkMin	1.84	1.EgfR.on 526.Ia5Ib1.on 16.Shc.by.EgfR 4.Cbl.by.EgfR 853.PI3K.by.Cbl 54.Ca++.by.Integrin 99.Fak.on 12.Plcg1.by.Fak 84.IP3.by.Plc 561.Cav1.by.Integrin
	zChaff	2.48	526.Ia5Ib1.on 54.Ca++.by.Integrin 584.Eif4e.xby.4Ebp1 1.EgfR.on 567.PKA.xby.Integrin 612.Eps8.by.Actin 99.Fak.on 12.Plcg1.by.Fak 84.IP3.by.Plc 498.E3b1.by.Eps8
	ICS	6.16	1.EgfR.on 761.Grb2.by.EgfR 353.Gab1.on 167.Shp2.by.Gab1 16.Shc.by.EgfR 526.Ia5Ib1.on 99.Fak.on 12.Plcg1.by.Fak 84.IP3.by.Plc 172.Shp1.by.EgfR
NuSMV	SIM	1.2	526.Ia5Ib1.on 99.Fak.on 12.Plcg1.by.Fak 84.IP3.by.Plc
Maude		<1	'1.EgfR.on '526.Ia5Ib1.on '54.Ca++.by.Integrin '4.Cbl.by.EgfR '870.Crk.by.Cbl '563.Dock.by.Crk '38.Elmo.by.Dock '99.Fak.on '761.Grb2.by.EgfR '353.Gab1.on '713.Ilk.by.Integrin '434.Pax.phos '440.Cas.on '853.PI3K.by.Cbl '322B.Eif4g.by.PI3K '567.PKA.xby.Integrin '12.Plcg1.by.Fak '84.IP3.by.Plc '314.cPkc.on '758.nPkc.on-1 '16.Shc.by.EgfR '172.Shp1.by.EgfR '167.Shp2.by.Gab1 '343.Sos1.on '561.Cav1.by.Integrin '256.Rac1.on-3 '473.Mekk1.by.Rac1 '148.Mek.by.Mekk1 '472.Mekk4.by.Rac1 '372.Mkk4.by.Mekk1 '566.Pak.on '711.Mkk3/6.by.Pak '61.P38.by.Mkk3 '448.P38.to.nuc '64.Elk1.by.P38 '263.Msk.by.P38 '442.Creb.by.Msk deadlock
Lola		–	–

Table A.8: "EgfDemo" dish – $\mathbf{F}(\text{IP3-CLm}) \Rightarrow \neg(\neg(\text{PIP3-CLm})\ \mathbf{U}\ \text{IP3-CLm})$

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		74.44	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 440.Cas.on 12.Plcg1.by.Fak 512.Crk.on 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3 472.Mekk4.by.Rac1 84.IP3.by.Plc 457.Mkk3.by.Mekk4 61.P38.by.Mkk3 448.P38.to.nuc 758.nPkc.on-1 54.Ca++.by.Integrin 263.Msk.by.P38 1.EgfR.on 16.Shc.by.EgfR 343.Sos1.on 442.Creb.by.Msk 241.Grb2.by.Shc 612.Eps8.by.Actin 713.Ilk.by.Integrin 314.cPkc.on 172.Shp1.by.EgfR 353.Gab1.on 498.E3b1.by.Eps8 458.Mkk4.by.Mekk4 4.Cbl.by.EgfR 566.Pak.on 370.Mekk1.by.Pak1 167.Shp2.by.Gab1 62.PI3K.by.Rac1 567.PKA.xby.Integrin 561.Cav1.by.Integrin 584.Eif4e.xby.4Ebp1 322B.Eif4g.by.PI3K 64.Elk1.by.P38 DEADLOCK
SAL-BMC	Siege	17.44	no counterexample between depths: [0, 10]
	BerkMin	54.23	no counterexample between depths: [0, 30]
	zChaff	239.88	526.Ia5Ib1.on 99.Fak.on 434.Pax.phos 440.Cas.on 512.Crk.on 563.Dock.by.Crk 38.Elmo.by.Dock 256.Rac1.on-3 1.EgfR.on 566.Pak.on 711.Mkk3/6.by.Pak 54.Ca++.by.Integrin 61.P38.by.Mkk3 448.P38.to.nuc 473.Mekk1.by.Rac1 263.Msk.by.P38 343.Sos1.on 372.Mkk4.by.Mekk1 761.Grb2.by.EgfR 442.Creb.by.Msk 148.Mek.by.Mekk1 612.Eps8.by.Actin 4.Cbl.by.EgfR 584.Eif4e.xby.4Ebp1 353.Gab1.on 853.PI3K.by.Cbl 472.Mekk4.by.Rac1 12.Plcg1.by.Fak 713.Ilk.by.Integrin 64.Elk1.by.P38 172.Shp1.by.EgfR 84.IP3.by.Plc 314.cPkc.on 561.Cav1.by.Integrin 16.Shc.by.EgfR 498.E3b1.by.Eps8 167.Shp2.by.Gab1 567.PKA.xby.Integrin 322B.Eif4g.by.PI3K 758.nPkc.on-1 DEADLOCK
	ICS	90.22	no counterexample between depths: [0, 10]
NuSMV	SIM	1.3	526.Ia5Ib1.on 99.Fak.on 12.Plcg1.by.Fak 84.IP3.by.Plc
Maude		<1	'1.EgfR.on '526.Ia5Ib1.on '54.Ca++.by.Integrin '4.Cbl.by.EgfR '870.Crk.by.Cbl '563.Dock.by.Crk '38.Elmo.by.Dock '99.Fak.on '761.Grb2.by.EgfR '353.Gab1.on '713.Ilk.by.Integrin '434.Pax.phos '440.Cas.on '853.PI3K.by.Cbl '322B.Eif4g.by.PI3K '567.PKA.xby.Integrin '12.Plcg1.by.Fak '84.IP3.by.Plc '314.cPkc.on '758.nPkc.on-1 '16.Shc.by.EgfR '172.Shp1.by.EgfR '167.Shp2.by.Gab1 '343.Sos1.on '561.Cav1.by.Integrin '256.Rac1.on-3 '473.Mekk1.by.Rac1 '148.Mek.by.Mekk1 '472.Mekk4.by.Rac1 '372.Mkk4.by.Mekk1 '566.Pak.on '711.Mkk3/6.by.Pak '61.P38.by.Mkk3 '448.P38.to.nuc '64.Elk1.by.P38 '263.Msk.by.P38 '442.Creb.by.Msk deadlock
Lola		–	–

Table A.9: "EgfDemo" dish – $\mathbf{F}$(IP3-CLm) $\Rightarrow \neg(\mathbf{G}(\neg$(PIP3-CLm)))

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		–	failed to timely terminate
SAL-BMC	Siege	48.93	992k.Rgs7.off 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 1014k.TLR6.TLR2.on 857k.Csf1R.on 928k.Rasa1.by.Csf1R 974k.IfnaR.on 910k.Pi3k.by.Csf1R 643k.PIP3.from.PIP2.by.Pi3k
	BerkMin	49.26	857k.Csf1R.on 858k.Fyn.by.Csf1R 741k.RxR.by.9cisRA 983k.Nos2agene.on 631k.Ifnbgene.on 939k.Shc.by.Csf1R#2 902k.PafR.on 904k.PafR.xby.Grk1 919k.Pi3k.by.Slk 643k.PIP3.from.PIP2.by.Pi3k
	zChaff	64.86	897k.P2Ry2.on 980k.LxR.on 926k.PtgeR3.on 96.Pi3k.by.Gbg 967k.IL10R.by.IL10 857k.Csf1R.on 991k.Pparg.on 741k.RxR.by.9cisRA 900k.IL1R1.by.IL1 643k.PIP3.from.PIP2.by.Pi3k
	ICS	–	failed to timely terminate
NuSMV	SIM	2.5	857k.Csf1R.on 910k.Pi3k.by.Csf1R 643k.PIP3.from.PIP2.by.Pi3k
Maude		<1	Over 100 transitions with tens of redundant transitions

Table A.10: “Kitano Macrophage” dish – **G**(¬(PIP3-CLm))

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		–	failed to timely terminate
SAL-BMC	Siege	50.86	282k.Edg2.on 181.Pi3k.by.Gq 236k.Grk2.on 963k.CcR2.on 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on
	BerkMin	54.59	857k.Csf1R.on 939k.Shc.by.Csf1R#2 991k.Pparg.on 980k.LxR.on 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on
	zChaff	96.74	179k.TgfbR1.on 999k.Smad3.on 299k.Smad3.to.nuc 840.Smad4.to.cyto 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on
	ICS	–	failed to timely terminate
NuSMV	SIM	3.5	741k.RxR.by.9cisRA 52.Elk1.by.Jnk 610k.Fosgene.on
Maude		<1	Over 100 transitions with tens of redundant transitions

Table A.11: “Kitano Macrophage” dish– **G**(¬(Fosgene-on-NUc))

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		–	failed to timely terminate
SAL-BMC	Siege	55.22	199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on
	BerkMin	53.72	857k.Csf1R.on 939k.Shc.by.Csf1R#2 991k.Pparg.on 980k.LxR.on 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on
	zChaff	95.17	179k.TgfbR1.on 999k.Smad3.on 299k.Smad3.to.nuc 840.Smad4.to.cyto 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on
	ICS	–	failed to timely terminate
NuSMV	SIM	–	failed to timely terminate
Maude		–	–

Table A.12: "Kitano Macrophage" dish – **AG**(¬(Fosgene-on-NUc))

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		–	failed to timely terminate
SAL-BMC	Siege	194.66	609k.Csf2R.on 993k.Pi3k.by.Csf2R 643k.PIP3.from.PIP2.by.Pi3k 108k.Pdk1.by.PIP3 111k.aPkc.on 179k.TgfbR1.on 999k.Smad3.on 299k.Smad3.to.nuc 300.Mycgene.inhib 539.Jaks.by.IfngR 540k.Stat1.by.IfngR 702k.TnfR1.by.Tnfa 1018k.Traf1.Traf2.reloc 894k.Tradd.by.TnfR1 316.Smad7.by.Cbpp300 896.Rip.by.Tradd 583.Nik.on.1 369.Mekk1.by.Nik 148.Mek.by.Mekk1 000k.Erk.on 282k.Edg2.on 438k.Elk1.by.Erk 610k.Fosgene.on 198.Smad7.to.CLi
	BerkMin	345.98	702k.TnfR1.by.Tnfa 894k.Tradd.by.TnfR1 896.Rip.by.Tradd 1018k.Traf1.Traf2.reloc 951k.Nik.on.3 369.Mekk1.by.Nik 148.Mek.by.Mekk1 000k.Erk.on 785k.Edg3.on 83k.Plcb.on 84k.IP3.by.Plc 869k.ItpR.open 321k.Ca2+.by.ItpR 96.Pi3k.by.Gbg 666k.Calm.on 26.Erk.xby.Dusp1 388k.Camk2.on 758k.nPkc.on-1 000k.Erk.on 925k.PtgeR2.on 510k.Pld.by.Pkc 838.Smad2.xby.Pkc 74k.Adcy2.on 438k.Elk1.by.Erk 741k.RxR.by.9cisRA 610k.Fosgene.on 980k.LxR.on
	zChaff	–	failed to timely terminate
	ICS	–	failed to timely terminate
NuSMV	SIM	9.4	702k.TnfR1.by.Tnfa 52.Elk1.by.Jnk 1018k.Traf1.Traf2.reloc 898.Mekk1.by.Gck 148.Mek.by.Mekk1 000k.Erk.on 610k.Fosgene.on
Maude		2.8	Over 100 transitions with tens of redundant transitions

Table A.13: “Kitano Macrophage” dish – **F**(Fosgene-on-NUc)⇒¬(Erk-act-CLi) **U** Fosgene-on-NUc

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		–	failed to timely terminate
SAL-BMC	Siege	52.12	899k.P2Ry6.on 181.Pi3k.by.Gq 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on 974k.IfnaR.on
	BerkMin	52.95	1001k.Srebf1.Scap.Insig 179k.TgfbR1.on 997.Smad2.on 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on 190.Smad157.to.nuc
	zChaff	100.95	980k.LxR.on 925k.PtgeR2.on 96.Pi3k.by.Gbg#1 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on 991k.Pparg.on
	ICS	–	failed to timely terminate
NuSMV	SIM	4.3	741k.RxR.by.9cisRA 52.Elk1.by.Jnk 610k.Fosgene.on
Maude		2.2	Over 100 transitions with tens of redundant transitions

Table A.14: "Kitano Macrophage" dish – **F**(Fosgene-on-NUc)⇒¬(¬(Erk-act-CLi) **U** Fosgene-on-NUc)

Tool	SAT solver	Time secs	Counterexample generated:
SAL-SMC		–	failed to timely terminate
SAL-BMC	Siege	72.27	741k.RxR.by.9cisRA 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on 857k.Csf1R.on 922k.Plcg.by.Csf1R 990k.Plcg.xby.Ptpn1
	BerkMin	66.73	980k.LxR.on 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on 857k.Csf1R.on 922k.Plcg.by.Csf1R 990k.Plcg.xby.Ptpn1
	zChaff	126.38	980k.LxR.on 199k.TLR3.on 200k.Trif.by.TLR3 206.Traf6.by.Trif 716.P38a.on 64.Elk1.by.P38 610k.Fosgene.on 857k.Csf1R.on 922k.Plcg.by.Csf1R 990k.Plcg.xby.Ptpn1
	ICS	–	failed to timely terminate
NuSMV	SIM	5.4	741k.RxR.by.9cisRA 52.Elk1.by.Jnk 610k.Fosgene.on
Maude		–	failed to timely terminate

Table A.15: "Kitano Macrophage" dish – **F**(Fosgene-on-NUc) ⇒ ¬(**G**(¬(Erk-act-CLi)))

Appendix B

New Actors integrated in IOP

B.1 Actors: Introduction

The actor model is a mathematical model of concurrent computation, initially proposed by Hewitt et al. in [39]. A wide range of computation paradigms can be expressed through the actor model, which *"directly supports encapsulation and sharing, and provides a natural extension of both functional programming and object style data abstraction to concurrent open systems"* [22].

Actors are *"self-contained, concurrently interacting entities of a computing system"* [22]. They communicate via asynchronous message passing. In response to a message that it receives, an actor can make local decisions, create more actors, send more messages, and designate how to respond to the next message received. It is worth mentioning that the pool of actors in a computing system can be dynamically created and its topology can change dynamically.

B.2 Integrating SAL in IOP

Our main idea was to make communication possible between SAL and formal reasoning tools like Maude and PVS, which are already integrated as actors in the Interoperability Platform [10]. The usual scenario would be, for example, Maude sending a query to SAL, the latter executes the query and returns the result back to Maude. In order to accomplish this, we had to take into account that IOP's design is based on the actor model of distributed computation [21].

IOP consists of numerous actors that can interact and interoperate with each other via asynchronous message passing, and some of them can even spawn other actors. The IOP registry or the System actor plays the role of a *"local post office"* [11]. An actor in IOP is a UNIX style process that has to be registered with the System actor according to a simple procedure described in [11]. Part of the registration process involves allocating three FIFO's, or UNIX style named pipes, and redirecting the actor's stdin, stdout, stderr file descriptors to these special files.

The tools built into SAL have different execution behaviours. The majority of them (i.e. sal-smc, sal-bmc, sal-path-finder, sal-deadlock-checker) evaluate a request and exit after returning a result. On the other hand, one tool called Sal Simulator (i.e. sal-sim) is a Scheme based system programmed as a "read-eval-print" loop. This means that sal-sim waits in a "read-eval-print" loop for a request, evaluates the request, returns a result and waits for another request. The Sal Simulator does not exit, unless it is terminated explicitly. Thus, we have adopted two different strategies for integrating SAL tools into IOP:

- For integrating the majority of SAL tools that exit after they return a result we have developed two SAL actors which consist of two distinct C programs:
  - A SalSpawner, which listens to clients actions and spawns by demand new actors called SalActors, which are enrolled with the System actor. The SalSpawner makes sure each SalActor has a unique name (usually the name is SalActor<index>), such that there are no name conflicts when registering the actor with the IOP registry.
  - A SalActor, which is composed of a parent and a child process. The child process is the actual SAL process. The parent process monitors the error and output stream of the child process via pipes and using a separate error thread. The parent process redirects to stderr and stdout all the messages that are printed by the SAL process to the error stream and respectively to the output stream. After the child process dies (i.e. the SAL process terminates), the parent process simply unenrolls itself from the IOP registry and terminates.
- For integrating the Sal Simulator into IOP we have developed a C application called SalWrapper, which is in fact another SAL actor. This actor consists of two processes:

the child process running sal-sim, and a parent or wrapper process acting as an interface for the SAL tool and the underlying message system. The parent process waits in an infinite loop for the user's input, echoes the input to the child/tool, and waits for a result from SAL. Similarly with the SalActor, the parent process within the SalWrapper redirects to stderr and stdout all the messages that are printed by the SAL process to the error stream and respectively to the output stream.

B.3 Creating and using the SAL actors

The SAL actors can be created or terminated either explicitly by sending the System actor a request to either start or stop the actor, or by describing the desired actors at startup in the .ioprc file, as described in the IOP manual [11]. We describe briefly a sample use of these two different kinds of SAL actors.

B.3.1 Sample use of SalSpawner and SalActor

The SalSpawner actor must be created in the first place. This is done by sending the IOP registry (or the System actor) the following message:

```
(user start salspawner iop_sal_spawner *FIFO_IN* *FIFO_OUT*)
```

To create different SAL actors, one must send the SalWrapper actor the following message repeatedly:

```
(user opensalactor)
```

Once this is done, the client can choose any SAL actor from IOP's drop-down list (they are indexed from 0,1,...) and send a request to the respective actor. Below are some sample requests that can be sent to sal-bmc, sal-smc, sal-path-finder and sal-deadlock-checker using the SAL actors:

```
(user sal-bmc peterson livenessbug1)
(user sal-smc peterson mutex)
(user sal-path-finder -d 100 peterson system)
(user sal-deadlock-checker peterson system)
```

After the SAL processes terminate, the respective SAL actors unenroll from the IOP registry and terminate automatically. In order to terminate and unenroll the SalSpawner actor from the IOP registry, the following request must be sent to the System actor:

```
(user stop salspawner)
```

B.3.2 Sample use of SalWrapper

To create the SalWrapper actor the following message must be sent to the System actor:

```
(user start SalWrapper iop_sal_wrapper)
```

Once this is done, the only SAL tool which can be tested using this actor is sal-sim. The reason for this has been explained above. Therefore, in order to send requests to the Sal Simulator, we need to first load its source, import a SAL context, start the simulation session and perform the actual requests. These steps must be performed in a strict order, as demonstrated below:

```
(sal/load-source! "sal-simulator-front-end.scm")
(import! "peterson")
(start-simulation! "system")
(step!)
(display-curr-trace)
(display-curr-states)
```

To exit the Sal Simulator we need to send the SalWrapper actor the following message:

```
(exit)
```

This will not only kill the SAL tool, but it will also terminate and unenroll the SalWrapper actor from the IOP registry.