Validating Specifications of Dynamic Systems Using Automated Reasoning Techniques[*]

Remco Feenstra[†] Roel Wieringa

Faculty of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
email: rbfeens@cs.vu.nl, roelw@cs.vu.nl

Abstract

In this paper, we propose a new approach to validating formal specifications of observable behavior of discrete dynamic systems. By *observable* behavior we mean system behavior as observed by users or other systems in the environment of the system. Validation of a formal specification of an informal domain tries to answer the question whether the specification actually describes the intended domain. This differs from the verification problem, which deals with the correspondence between formal objects, e.g. between a formal specification of a system and an implementation of it. We consider formal specifications of object-oriented dynamic systems that are subject to static and dynamic integrity constraints. To validate that such a specification expresses the intended behavior, we propose to use a tool that can answer reachability queries. In a *reachability query* we ask whether the system can evolve from one state into another without violating the integrity constraints. If the query is answered positively, the system should exhibit an example path between the states; if the answer is negative, the system should explain why this is so. An example path produced by the tool can be used to produce scenarios for presentations of system behavior, but can also be used as a basis for acceptance testing. In this paper, we discuss the use of planning and theorem-proving techniques to answer such queries, and illustrate the use of reachability queries in the context of information system development.

1 Introduction

When modeling an informal domain using a formal specification language, for example during development of an information system, it is often necessary to check whether the result-

[*]This research is partially supported by the the Esprit Basic Research Action IS-CORE (Information Systems—COrrectness and REusability) (working group 6071). This paper is an adapted, shortened version of our presentation at the workshop on Semantics in Databases, Řež, Czech Republic, January 1995.

[†]This research is partially supported by the Netherlands Organization for Scientific Research (NWO), SION project 612–317–408.

ing specification (the *Conceptual Model* (CM)) reflects this domain faithfully enough to be usable. The activity of checking this correspondence between the informal domain and the formal specification is called **validation**.

In the case the modeling is done in the initial stages of a development process, the modeled system does not yet exist in reality: it "exists" only informally in the minds of domain specialists. This situation precludes direct empirical validation of the CM. Instead, an analyst needs to adopt an indirect approach by soliciting comments from domain specialists as to whether the formal CM specification conforms to their ideas about the system. When trying to do so, the analyst encounters the problem that domain specialists with little background in formal specification often don't understand the formal CM specification. When the formal meaning of the specification differs from the analyst's idea of what was specified, for example due to mistakes, the analyst might misunderstand the formal meaning of the specification as well.

Studying concrete examples of the specified behavior can help in understanding and discussing the specification. In the case of a system model based on states and transitions, it is useful to present examples of system states, and be able to simulate the effect of selected transitions by showing the resulting state after the transition. This approach is convenient when the formal specification is executable in the sense that we can compute the effects of a transition on a state, for example when we expressed our specification in an executable specification language. We may enhance the presentation by embellishing the runs of such a simulation with **explanations** of the runs in terms of the CM. We call such explained runs of the system an **animation** of the system. The explanation of a system run might take the form of a nice graphical presentation of the run. For example, object behavior may be represented by pictures of finite state automatons, and object state may be represented in tabular or other form. This way, an prototype-based animator resembles a graphical debugger for formal specifications, that repeatedly shows the effects of a transaction selected by the analyst.

In this paper, we propose an approach to animation of specifications of state-transition based systems that can supplement and enhance the power of graphical animations. We argue that a system for creating animations capable of *reasoning about reachability properties* of system states would reduce two common problems of the simulation-based approach to animation mentioned above, viz.

(1) *(Relevance of information)* A snapshot of a system state of a system of any complexity shows typically many irrelevant details that distract the viewer from the properties he should watch carefully. The analyst can reduce the number of irrelevant details by instructing the animation system what to show and what not, but this requires so much effort that it is not practical.

(2) *(Reasoning about scenarios)* When investigating the state space of the specified system by repeatedly executing transactions of the system on demand, and observing their effects on the system state, the analyst must still manually provide suitable starting states and select transactions that demonstrate interesting behavior. Again, this requires much effort. In addition, it may cause the analyst to ignore some examples of behavior that can occur according to the specification, but not according to the analyst's intuition of the system.

In our approach, instead of basing an animation on showing the effects of transitions selected by the analyst on some example state, the analyst formulates properties of interesting behavior in terms of queries about the reachability of system states from other states under constraints. Software capable of automatically solving these queries will then generate examples of states and transactions that demonstrate behavior exhibiting these properties. This reduces problem (1) by enabling the animation system to derive relevance judgments about the information to be presented in the presentation, because it knows more of the intended context via the query. Problem (2) is reduced because more of the reasoning involved in finding examples of interesting behavior is automated.

When we are not willing to restrict our specification language severely, solving general reachability queries is undecidable. Fortunately, in the context of validation of practical systems like database systems, we can make some reasonable assumptions such that techniques from planning and model generation become applicable. Although we cannot expect the animation tool to be able solve general reachability queries, let alone within reasonable time, a tool that can assist in simple cases occuring in practice can still be very helpful to an analyst.

In section 2, we sketch an example specification of a simple library in situation calculus, and discuss interesting reachability queries about it. Reachability questions are relevant to any system based on states and transitions. For example, reachability properties were studied thoroughly in Petri net theory [Rei85]. In a logic-based context they are less well-studied. Nevertheless, there are other sources for inspiration, such as plan generation techniques from AI and theorem-proving techniques for first-order logic. Section 3 discusses the ideas that we have borrowed from these fields in our approach to solving reachability queries. In addition, section 3 gives an outline for a procedure for solving reachability queries. The research reported on in this paper is still in the starting stage; in section 4 we discuss the feasibility of our approach for the improvement of animation techniques, and list some topics for current and further research.

2 Reachability queries

As an example, we will consider a very simple specification of the administration of a library in situation calculus [MH69]. In practice, an analyst would use a more convenient language for such a project such as LCM [FW93, Wie91], which is a syntactically sugared version of Dynamic Object Logic (DOL) [WJS94]. We assume that we already have a specification of the necessary data types and predicates on them available, including types consisting of an infinite supply of values that serve as object identifiers, which are globally unique proper names for the objects about we want to reason, such as library members, copies of books etc.

To keep track of the system dynamics, the specification uses state-dependent predicates, e.g. $\mathsf{Borrowing}$. The term $\mathsf{Borrowing(m135,\ c3,\ s)}$ might now represent the property that in the state of the world s, the library member with identifier $\mathsf{m135}$ has borrowed a copy with identifier $\mathsf{c3}$.

Static integrity constraints express regularities in the domain that any state of the model should satisfy. For example, we might express the required property that in any state of the world a copy of a book can be borrowed by at most one member at a time by the formula

$$\begin{aligned}&\forall \mathsf{s} : \mathsf{STATE}\ \forall \mathsf{c} : \mathsf{COPY}\ \forall \mathsf{m1,m2} : \mathsf{MEMBER}\\ &\quad \mathsf{Borrowing(m1,c,s)} \wedge \mathsf{Borrowing(m2,c,s)} \rightarrow \mathsf{m1} = \mathsf{m2}\end{aligned}$$

Then we might introduce terms for a fixed set of library transactions, that correspond, in the style of JSD [Jac83], to events in the universe of discourse, like the binary transaction borrow. A term borrow(m3,c4) might represent the occurrence of the event where a member with identifier m3 borrows a copy with identifier c4. Similarly, we might introduce the transactions reserve and get_reserved_copy to represent the events of placing a reservation of a copy and getting the copy when it becomes available. To capture the effects of transactions, we include **transaction effect axioms** like

$$\forall \mathsf{s} : \mathsf{STATE}\ \forall \mathsf{m} : \mathsf{MEMBER}\ \forall \mathsf{c} : \mathsf{COPY}$$
$$\mathsf{Possible(borrow(m,c),s)} \longrightarrow \mathsf{Borrowing(m,c,do(borrow(m,c),s))}$$

that relate properties of different states of the world. In this paper, we will consider deterministic transactions only, i.e. from one state, two executions of the same transaction will always lead to the same result state.

The applicability of a transaction in a state is expressed by **transaction precondition axioms**, that state necessary preconditions for success of the transaction. For example, the requirement that a borrow transaction may only occur if no one has borrowed the copy at the moment might be expressed as

$$\forall \mathsf{s} : \mathsf{STATE}\ \forall \mathsf{m,n} : \mathsf{MEMBER}\ \forall \mathsf{c} : \mathsf{COPY}$$
$$\mathsf{Possible(borrow(m,c),s)} \longrightarrow \neg \mathsf{Borrowing(n,c,s)}$$

Implicitly, we add the **frame assumption**, that augments the specification of action effects to the effect that apart from the changes described by the transaction effect axioms, nothing else changes as an effect of a transaction. Additionally, for reasons of representability of states, we add the requirement that in every state of the world, the extension of all state-dependent predicates is finite. This also applies to the special state-dependent predicate Exists, which keeps track of which of the possible object instances do actually exist in a particular state of the world.

Informally, in the intended model of the specification, a state s_2 can be reached from another state s_1 via a transaction t, written as $s_1 \xrightarrow{t} s_2$, iff all preconditions for t are satisfied in s_1, both states satisfy the static integrity constraints, and the transition corresponds to a minimal change respecting the transaction effect axioms, finiteness and frame assumption. Note that the static integrity constraints only restrict the applicability of transactions; they do not cause derived updates.

A **scenario** is a particular finite "run" in of the specified system. More formally, it is a finite sequence $\mathbf{s} = s_0, s_1, \ldots, s_n$ of states s_i and a finite sequence $\mathbf{p} = t_0, t_1, \ldots, t_{n-1}$ of transitions t_i for some integer $n \geq 0$, such that $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} s_n$.

A **reachability query** asks for a scenario that is an example of behavior of the specified system that satisfies additional constraints. A general reachability query asks for an example scenario that demonstrates the truth of an existentially qualified formula, as below, in the intended model of the specification:

$$\exists n \geq 0 \exists \mathbf{s} = \langle s_0, s_1, \ldots, s_n \rangle \exists \mathbf{p} = t_0; t_1; \cdots; t_{n-1}\ s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} s_n \wedge \Phi(\mathbf{s}, \mathbf{p})$$

In this formula, $\Phi(\mathbf{s}, \mathbf{p})$ is a situation calculus formula that expresses constraints on the states and transactions occuring in the scenario. Again, in practice the analyst would prefer to use a more convenient language for expressing reachability queries, perhaps based on dynamic logic, but we will stick to situation calculus here.

Examples of validation questions about our library specification that are easily formulated as reachability queries include the question whether there is a possible system state at all, expressed as the degenerate query $\exists n \geq 0 \exists \mathbf{s} \exists \mathbf{p} true$. This amounts to checking whether the static integrity constraints are consistent.[1] In our case, the system might generate the scenario for $n = 0$, consisting of a single state s_0, in which the extension of all state-dependent predicates is empty. As this includes the Exists predicates, no objects exists in this state. Thus we can be sure that our system has at least this state. Thus, we have shown the formal property of consistency of the specification, but the informal property as whether this "empty" state is also considered a valid example state of a library system by the domain specialists is a different matter. This can be tested by presenting this example to them and asking whether they agree that this is what they intended.

Queries for the existence of system states with certain properties, and for examples of the possibility of occurrence of a transaction are easily expressed as well.

As a more complicated example, consider the case that, according to domain specialists, a member who has an outstanding fine should not be able to obtain more copies until he has paid the fine. We would like to know whether this property also holds in the specified system. We will try to refute this property by asking for a scenario in which a member with an outstanding fine is able to arrive in a state where he has borrowed another copy, without paying the fine first:

$$\begin{aligned} &\exists n \geq 0 \exists \mathbf{s} = \langle s_0, s_1, \ldots, s_n \rangle \exists \mathbf{p} = t_0; t_1; \cdots t_{n-1} \exists \mathsf{m} : \mathsf{MEMBER}\, \exists \mathsf{c} : \mathsf{COPY} \\ &\quad \mathsf{Exists}(\mathsf{m}, s_0) \wedge \mathsf{MemberFine}(\mathsf{m}, s_0) \wedge \neg \mathsf{Borrowing}(\mathsf{m}, \mathsf{c}, s_0) \\ &\quad \wedge \mathsf{Borrowing}(\mathsf{m}, \mathsf{c}, s_n) \wedge \neg \mathsf{Occurs}(\mathsf{pay}(\mathsf{m}), \mathbf{p}) \end{aligned}$$

Perhaps to our surprise the system might answer that there *is* a scenario that satisfies these constraints. Because, accidentally, the preconditions of reserve and get_reserved_copy transactions omit the check on outstanding fines, a member can still obtain more copies while having an outstanding fine, by a back door approach: first reserve them and then get the reserved copies. Whether this was intended by the domain specialists, and how to modify it if not, can then be discussed with the domain specialists showing them this example scenario and asking for their comments.

3 Solving reachability queries

In their general form, answering reachability queries automatically is out of question because this problem is undecidable for sufficiently powerful specification languages. We can, however, still try to solve interesting sub cases by automated reasoning techniques. We will first discuss the relation of the problem of solving reachability question with the problem of plan generation, which has received much attention in the AI literature. Then we describe how to adapt planning techniques for solving reachability queries.

3.1 Plan generation versus solving reachability queries

The problem of generating plans to achieve goals bears resemblance to the problem of generating scenarios as solutions to reachability queries. When we look in more detail, however, we can also see some important differences.

[1]Although this is usually described as a formal property of the specification, it is a validation question as well: if the specification has no models at all, it certainly can't describe the intended system.

The plan generation problem from AI[2] can be formulated as finding a value for a plan variable **p**, consisting of a finite sequence of actions, demonstrating the truth of an existentially quantified formula of the form

$$\exists n \geq 0 \exists \mathbf{p} = t_0; t_1; \ldots; t_{n-1} \forall \mathbf{s} = \langle s_0, s_1, \ldots, s_n \rangle$$
$$(s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} s_n \wedge \Phi_{\text{initial}}(s_0)) \rightarrow \Phi_{\text{final}}(s_n)$$

Compare this with finding scenarios as solutions to reachability queries, expressed as the problem of finding states **s** connected by actions **p**:

$$\exists n \geq 0 \exists \mathbf{s} = \langle s_0, s_1, \ldots, s_n \rangle \exists \mathbf{p} = t_0; t_1; \cdots; t_{n-1}$$
$$s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} s_n \wedge \Phi_{\text{initial}}(s_0) \wedge \Phi_{\text{final}}(s_n)$$

In plan generation and the similar problem of program synthesis, our primary interest is in obtaining a plan or program that will always lead to a state satisfying Φ_{final}, when started in some state satisfying Φ_{initial}, whereas in scenario generation we are interested in a particular execution of a plan or program. Nevertheless, some planning techniques can be used to solve reachability queries, as will be argued next.

3.2 Planning techniques

Solving reachability queries involves reasoning about actions, their applicability and their effects. These themes were discussed extensively in the AI literature about problem solving, especially in the context of generating plans to achieve goals. Green [Gre69] showed that many of these problem solving tasks can be tackled by formulating them in first-order logic using answer terms, and then applying a (resolution-style) general theorem proving program. As a side-effect of the search for a proof of the existence of a goal state, an answer is constructed in these answer terms, that is communicated to the user if the attempt succeeds.

The direct applicability of this technique is seriously hampered in practical situations by the frame problem [MH69]. Finding ways of dealing efficiently with the frame assumption has subsequently become one of the central topics in planning, with landmark systems such as STRIPS [FN71] that exploit make efficient reasoning about action effects possible, although they impose severe restrictions on the specification of action effects.

Of our particular interest is a proposal by Reiter [Rei91], who combines proposals by Pednault [Ped89] and Schubert [Sch90] into a convenient specification language of action effects in a subset of situation calculus. It is based on supplying a **successor state axiom** for each state-dependent predicate P and the **specification completeness assumption** that these successor state axioms capture all conditions leading to change of the updatable predicates. Assuming that we can express a sufficient precondition for success of the transaction t in a state s as a formula (abbreviated here as $\mathsf{SPFS}(t,s)$), the successor state axiom for the state-dependent predicate P has the form

$$\forall t : \mathsf{TRANSACTION}\ \forall s : \mathsf{STATE}\ \mathsf{SPFS}(t,s) \rightarrow (\forall \mathbf{x}\, P(\mathbf{x}, \mathsf{do}(t,s)) \leftrightarrow \Psi_P(\mathbf{x},t,s))$$

[2]We will only consider the elementary problem of generating unconditional plans and assume that the world can only change due to the execution of the plan.

where $\Psi_P(\mathbf{x}, t, s)$ is a formula containing no applications of do. For instance, it might have the form $\Psi_P(\mathbf{x}, t, s) \equiv \gamma_P^+(\mathbf{x}, t, s) \vee P(\mathbf{x}, s) \wedge \neg\gamma_P^-(\mathbf{x}, t, s)$ where the formula $\gamma_P^+(\mathbf{x}, t, s)$ exhaustively expresses all conditions that can lead to P becoming true after execution of transaction t, and $\gamma_P^-(\mathbf{x}, t, s)$ all conditions that cause P to become false after execution of t. Both are expressed in terms of conditions on state s.

Now we can use regression in planning to reason efficiently about action effects, which essentially works by computing explicitly formula's that express preconditions that must be met for properties to hold after a transaction.

The generation of a plan of actions leading from state s_1 to s_2 is now reduced to two subproblems: (1) search for sequences of actions that could lead from s_1 to s_2, generating a condition for applicability of this plan in terms of properties of s_1 by means of regression; and (2) test whether the description of s_1 implies this condition. This is a standard task for an automatic theorem prover.

Conversion of our specification to Reiter's specification format might be very reasonable in practical system modeling context, like when modeling a database system. We need to make the following assumptions:

- To be able to formulate action effects as successor state axioms, we need to have complete knowledge of the action effects. Although this is a problem in general, in many practical modeling contexts this assumption is satisfied. This is because, in practice, we already need to bound and simplify our domain anyway to obtain a model. Moreover, the action effect specification has to be in a format that allows us to derive the successor state axioms from it automatically.

- We have complete knowledge of action preconditions (the qualification problem) in order to build sufficient conditions for success. For the same reason as above, this assumption is satisfied. We can easily specify them by writing a specification that mentions only the necessary preconditions for success, and making the assumption that the specification mentioned all of them. Then the sufficient precondition for success of an action is the conjunction of all these necessary preconditions for its success.

- We have no derived updates due to constraints. This is a restriction that may have to be lifted later.

- We have no disjunctive information in states. Again, this is a restriction that may have to be lifted later.

3.3 Construction of example states

Reiter's representation and regression discussed above allow us to reduce problem of planning to theorem proving. The same approach could be applied for solving reachability queries too, if we modify it to reflect the way it differs from planning in the way quantification over states is done, as discussed in section 3.1. This modification consists of replacing the theorem prover, used to test whether the initial state formulas imply the regressed goal formula, by a component that tests the satisfiability of the conjunction of the initial state formulas and the regressed goal formula. In the case of satisfiability, it should also construct a model that demonstrates this. How can such a component be built?

Methods for testing the *satisfiability* of a set of formulas are scarcely found in the automated reasoning literature. Instead, most approaches aim at showing the *validity* of a formula, often by means of showing the inconsistency of its negation. Sometimes, we can get a proof of satisfiability as spin-off of a failed attempt to prove inconsistency. This is for example the case in the method of analytic tableaux [Smu68], when a path in the proof tree is encountered that cannot be extended any further by the tableau extension rules, but doesn't contain a contradiction. The formulas on such a path constitute a Hintikka set, corresponding to a model for the formula. Caferra [Caf93] tried to integrate the systematic search for models into the tableau method by means of techniques for solving equational problems. Similar proposals exist for resolution-based methods.

In their original form, these methods are not directly applicable to our problem, because of our additional (non-first-order) requirement on our intended model that all extensions of state dependent predicates in them must be finite. This is related to the problem of finding models with a finite number of elements of first-order formulas.

The method of analytical tableaux, nor its extension by Caferra are complete for finding finite models. This is caused by the liberal introduction of constants of each sort by the tableau extension rules for quantifiers. For example, the tableau extension rule for a formula of the form $\exists x \phi$ introduces a fresh constant and requires $\phi[a/x]$ to hold. The problem is that it does so even if another already introduced constant would do as well.

Kung [Kun85] noted this problem when applying the tableaux method for proving the consistency of database specification, and proposed a variant of the tableaux method that restricts the introduction of new constants. Although this idea was good, his method is still not complete for finite satisfiability.

Bry and Manthey [BM86] analyzed the problem of semideciding finite satisfiability and unsatisfiability of integrity constraints. They describe how case analysis of models with different function evaluations can be added to the tableaux method to solve this problem.

Of related interest is the SATCHMO theorem proving system [MB86] by the same authors, that tries to prove theorems by failure of a systematic attempt to create concrete models of them, exploiting range-restrictedness in clauses. In its original form, SATCHMO is not complete for finite satisfiability, but Bry, Decker and Manthey [BDM88] apply a variant of it to test constraint satisfiability in a database context, where constraints are formulas with only restricted quantification. Constraints with restricted quantification arise quite naturally in our framework, by our separation of the set of possible object identifiers from the set of actually existing ones in a state, by means of the Exists predicate. Combined with the order-sorted approach of specifying a separate sort for each class and the fact that integrity constraints usually refer only to the existing objects only, range restrictedness is a reasonable assumption in our context. Although concrete database states are constructed in their paper, they seem to serve only as a temporary result of an approach to give a yes/no answer to finite satisfiability questions. Of course, in our intended application, we are very much interested in these models themselves.

3.4 Integration of techniques for solving reachability queries

Given the techniques discussed above, we are now in a position to outline an approach for solving reachability queries: (1) transform the specification into Reiter's format with successor state axioms and sufficient preconditions for success; (2) use search and regression of the goal formulas of the reachability query, obtaining as a subproblem a test for finite (un)satisfiability of a set of formulas relating to a single starting state; (3) apply the adapted version of the tableau method for testing finite satisfiability of a set of range-restricted constraints as described by Bry, Decker and Manthey; (4) when a starting state has been found that satisfies the integrity constraints and the regressed goal constraints, apply progression (computation of the successor states) on the generated action sequence to obtain the concrete intermediate and final states of the scenario.

Our prototype currently implements the second and third step; the first step is still performed manually, but work is being done to automate translation from the LCM specification language to reachability calculus. Currently we are also working on the implementation of data type handling (both evaluation and constraint solving) that is needed for the third and fourth step.

4 Discussion and future work

We believe that the approach outlined in this paper is feasible and would yield a useful addition to animation techniques. In the introduction, we mentioned two problems with traditional approaches to animation. The first problem is that there is a mass of irrelevant details to be suppressed during the animation. By answering reachability questions by means of theorem-proving and planning techniques, the system has, at least in principle, the information available that allows it to relate the (non)existence of a path from one state to another to the axioms in the specification. It can therefore (again at least in principle) focus on the presentation of the relevant aspects of the animation.

The second problem mentioned in the introduction is that in order to find interesting animations, the analyst has to perform much error-prone manual formal reasoning about the specification. Obviously, this problem would be avoided by our approach to animation.

Needless to say, there is still a considerable amount of research to be done before an animation system such as proposed in this paper is implemented. In order to further work out the ideas presented in this paper, we will build a small prototype based on a model-generation theorem prover like SATCHMO and work through a number of small examples with it to study the feasibility of animation based on reachability queries.

References

[BDM88] François Bry, Hendrik Decker, and Rainer Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 488–505, Venice, 1988. Springer-Verlag.

[BM86] François Bry and Rainer Manthey. Checking consistency of database constraints: a logical basis. In *Proc. 12th International Conference on Very Large Data Bases*, pages 13–20, Kyoto, August 1986.

[Caf93] Ricardo Caferra. A tableaux method for systematic simultaneous search for refutations and models using equational problems. *J. Logic Comput.*, 3(1):3–25, 1993.

[FN71] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3 and 4), 1971.

[FW93] R.B. Feenstra and R.J. Wieringa. LCM 3.0: a language for describing conceptual models. Technical Report IR-344, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December 1993.

[Gre69] C. Green. Application of theorem proving to problem solving. In *Proc. 1st IJCAI*, pages 219–239, 1969.

[Jac83] M. Jackson. *System Development*. Prentice-Hall, 1983.

[Kun85] C. Kung. A tableaux approach for consistency checking. In A. Sernadas, J. Bubenko Jr., and A. Olivé, editors, *Information Systems: Theoretical and Formal Aspects*, pages 191–207. Elsevier Science Publishers (North-Holland), 1985.

[MB86] Rainer Manthey and François Bry. SATCHMO: A theorem prover in PROLOG. In Jörg H. Siekman, editor, *Proceedings CADE-8*, 1986.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Ped89] E.P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R.J. Brachman, H. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332. Morgan Kaufmann Publishers, 1989.

[Rei85] W. Reisig. *Petri Nets*. Springer, 1985.

[Rei91] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press, 1991.

[Sch90] Lenhart Schubert. Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In Henry E. Kyburg, Ronald P. Loui, and Greg N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*. Kluwer, 1990.

[Smu68] R. Smullyan. *First-order Logic*. Springer-Verlag, 1968.

[Wie91] R.J. Wieringa. A formalization of objects using equational dynamic logic. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *2nd International Conference on Deductive and Object-Oriented Databases (DOOD'91)*, pages 431–452. Springer, 1991. Lecture Notes in Computer Science 566.

[WJS94] R.J. Wieringa, W. de Jonge, and P.A. Spruit. Roles and dynamic subclasses: a modal logic approach. In M. Tokoro and R. Pareschi, editors, *Object-Oriented Programming, 8th European Conference (ECOOP'94)*, pages 32–59. Springer, 1994. Lecture Notes in Computer Science 821.