Combining WS1S and HOL

David Basin and Stefan Friedrich
Institut für Informatik, Universität Freiburg
Am Flughafen 17, D-79110 Freiburg, Germany
{basin,friedric}@informatik.uni-freiburg.de

Abstract

We investigate the combination of the weak second-order monadic logic of one successor (WS1S) with higher-order logic (HOL). We show how these two logics can be combined, how theorem provers based on them can be safely integrated, and how the result can be used. In particular, we present an embedding of the semantics of WS1S in HOL that provides a basis for coupling the MONA system, a decision procedure for WS1S, with an implementation of HOL in the ISABELLE system. Afterwards, we describe methods that reduce problems formalized in HOL to problems in the language of WS1S. We present applications to arithmetic reasoning and proving properties of parameterized sequential systems.

1 Introduction

We investigate the combination of two logics and the integration of two powerful and complementary theorem proving systems for these logics:

- a decision procedure for WS1S (the weak monadic second-order logic of one successor) implemented in the MONA system, and
- an implementation of HOL (higher-order logic) in the ISABELLE system.

There are compelling reasons for investigating this combination. WS1S belongs to a class of monadic logics that are among the more expressive decidable logics known and many decision problems can be embedded in WS1S [Thomas, 1990]. The logic is also well suited for reasoning about many kinds of systems that can be modeled using automata [Basin and Klarlund, 1998]. However, as with all decidable logics, its expressiveness is limited. In contrast, HOL is a very expressive foundation for reasoning about programs and systems; however proof construction in HOL typically requires considerable user guidance. A combination potentially offers the complementary advantages of both: the expressive logic of HOL can be used to specify problems and MONA used to automatically solve subproblems expressible in WS1S that arise during HOL proofs.

In this paper, we examine the problems that arise in carrying out such a combination and propose solutions. First, the combination must be sound: only valid HOL formulae should be provable. Semantic methods are required here since the decision procedure for WS1S is a semantic one; the validity of WS1S formulae is determined by translating them to automata that recognize models. We show how a semantic embedding of WS1S can be used to guarantee the correctness of the combination. Within a conservative extension of HOL with theories of sets, numbers, and arithmetic, we formalize the semantics of WS1S formulae. WS1S formulae then correspond to a subclass of HOL formulae and MONA can be used to determine the validity of problems in this class.

Second, the combination must be usable: We require a way of generating WS1S subproblems during proofs in HOL, which can be solved by MONA. In our case, this is equivalent to generating problems in the range of our semantic embedding. There cannot be a general method of doing this, since that would give us a decision procedure for HOL, which is impossible. However, we show that for two particular problem domains such methods are possible. First, we show how to solve different classes of *linear arithmetic* problems in HOL. Linear arithmetic is the theory consisting of formulae made up from numeric constants, variables, addition, arithmetic relations ($<$, $\leq$, $=$), and the standard first-order logical connectives. We present decision procedures, and their integration in HOL, based on the linear arithmetic theories of the natural numbers and integers.

The second method is rather different and more difficult to characterize crisply. The problem domain we examine is reasoning about the correctness of parametric sequential circuits, e.g. a family of n-bit counters, which is parameterized by the number of bits n and its behaviour is a function of time. Such circuits cannot be directly formalized in WS1S or in any decidable logic.[1] Such problems can, however, be formalized in HOL and we show that it is possible to eliminate one of the two unbounded parameters in cases where the goal can be reduced to one involving only finitely many instances of this parameter. The reduced problem is expressible in WS1S whereby MONA can be used to establish properties that are invariant over time.

The remainder of the paper is organized as follows. In Section 2 we provide background both on ISABELLE's implementation of HOL, and on WS1S and its implementation in MONA. In Section 3 we describe the combination, its mechanization in HOL, and its correctness. Then, we consider two rather different applications: In Section 4 we show how to automate arithmetic rea-

[1]The reason for this is that the two parameters (bit-width and time) are independent. Such problems correspond, abstractly, to problems on grids, which are undecidable.

soning over the natural numbers and integers, and in Section 5 we present applications to verifying parameterized sequential hardware. Finally, in Section 6, we compare with related work and draw conclusions.

2 Background

2.1 Isabelle/HOL

We use ISABELLE's implementation of HOL for our work; integration with other theorem provers, such as the HOL system or PVS, would be similar.

ISABELLE [Paulson, 1994] is a generic theorem prover in which *object logics* are implemented in ISABELLE's *metalogic*, which is a fragment of intuitionistic higher-order logic based on the (polymorphically) typed lambda-calculus with polymorphic types. The logical connectives of this fragment are implication (written $\Longrightarrow$), universal quantification (written $\bigwedge$), and equality (written $\equiv$). Object logics are encoded by declaring a *theory*, which is a signature and a set of axioms. Theorems are constructed interactively by applying tactics (programs that construct proofs) in the metalogic. Theorems may also be imported from external theorem provers, which can serve as oracles for particular classes of problems.

Terms in HOL are constructed from the λ-calculus and three additional operators: implication $\rightarrow$, equality $=$, and Hilbert's description-operator ϵ. All terms are typed. For example, for x of type α, $\epsilon x.\, P(x)$, is a term of type α; this term denotes some a, for which $P(a)$ holds, if such an a exists, and an arbitrary term of type α otherwise (types are non-empty). As is standard in HOL, all other logical connectives are defined using these primitives, e.g., $\exists\, x.\ P(x) \equiv P(\epsilon x. P(x))$. There is no particular symbol for bi-implication; this is expressed using equality.

The initial theory of HOL is minimal and contains only 7 axioms. ISABELLE supports building hierarchies of theories based on (definitional) extensions with new types and constants. To ensure consistency, new inference rules are *derived*, e.g., the standard introduction and elimination rules for $\exists$ are derived from the above definition. A number of extensions come distributed with HOL. Important for our work are theories of typed sets, typed finite sets, natural numbers, and integers. Finite sets and natural numbers are inductively defined. Such inductive types come with induction principles and standard functions (e.g., addition, multiplication, and proper subtraction) are defined by primitive recursion. The integers are defined as equivalence classes over pairs of natural numbers in the standard way. We will say more about these data-types and their accompanying theories as we use them below.

2.2 WS1S and MONA

Research on WS1S and related monadic theories goes back to Büchi [Büchi, 1960] and Elgot [Elgot, 1961]. We briefly review syntax, semantics, and decidability.

Definition 1
Let x and X range over disjoint sets V_1 and V_2 of (first and second-order) variables. The language of WS1S is described by the following grammar.

$$\begin{array}{lcl} t & ::= & x \mid \mathsf{0} \mid \mathsf{s}(t) \\ \phi & ::= & t = t \mid t \in X \mid \phi \wedge \phi \mid \neg\phi \mid \exists x : \phi \mid \exists X : \phi \end{array} \qquad \blacksquare$$

Hence terms are built from first-order variables, the constant 0, and the successor symbol. Atomic formulae are built from the equality and membership predicates and formulae consist of atomic formulae and are closed under conjunction, negation, and quantification over first and second-order variables. Other connectives and quantifiers can be defined using standard classical equivalences, e.g., $\forall x : \phi \equiv \neg\exists x : \neg\phi$; the symbol $\{\!|\!\}$ represents the empty set. Also, it is possible to represent propositional variables and quantification over them (e.g. the propositional variable P can be represented by the atomic formula $\mathsf{0} \in X_P$, where X_P is a new second-order variable).

In the semantics of WS1S, formulae are interpreted in $\mathbb{N}$, where $\mathsf{0}$ and s are zero and the successor function, $=$ is equality over numbers, and $\in$ is membership. The logic is 'weak' because second-order variables are interpreted over *finite* subsets of the domain. The following embedding makes this interpretation clear. Here nat, finnat, and bool are the HOL types of naturals, finite sets of naturals, and truth-values respectively (i.e., relations are formalized as boolean-valued functions) and Suc denotes the successor function on natural numbers. In the sequel we shall omit typing constraints if types are known from the context.

Definition 2
The *embedding function* $\lceil \_ \rceil$ from WS1S formulae to HOL formulae is recursively defined as follows:

$$\begin{array}{rcll} \lceil x \rceil & = & x :: \mathsf{nat} & \text{if } x \in \mathrm{V}_1 \\ \lceil X \rceil & = & X :: \mathsf{finnat} & \text{if } X \in \mathrm{V}_2 \\ \lceil \mathsf{0} \rceil & = & \mathsf{0} :: \mathsf{nat} & \\ \lceil \mathsf{s}(t) \rceil & = & (\mathsf{Suc} :: \mathsf{nat} \Rightarrow \mathsf{nat})(\lceil t \rceil) & \\ \lceil t_1 = t_2 \rceil & = & (= :: [\mathsf{nat}, \mathsf{nat}] \Rightarrow \mathsf{bool})(\lceil t_1 \rceil)(\lceil t_2 \rceil) & \\ \lceil t \in X \rceil & = & (\in :: [\mathsf{nat}, \mathsf{finnat}] \Rightarrow \mathsf{bool})(\lceil t \rceil)(\lceil X \rceil) & \\ \lceil \phi_1 \wedge \phi_2 \rceil & = & (\wedge :: [\mathsf{bool}, \mathsf{bool}] \Rightarrow \mathsf{bool})(\lceil \phi_1 \rceil)(\lceil \phi_2 \rceil) & \\ \lceil \neg\phi \rceil & = & (\neg :: \mathsf{bool} \Rightarrow \mathsf{bool})(\lceil \phi \rceil) & \\ \lceil \exists x :: \phi \rceil & = & (\exists :: (\mathsf{nat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{bool})(\lambda x :: \mathsf{nat}. \lceil \phi \rceil) & \\ \lceil \exists X : \phi \rceil & = & (\exists :: (\mathsf{finnat} \Rightarrow \mathsf{bool}) \Rightarrow \mathsf{bool})(\lambda X : \mathsf{finnat}. \lceil \phi \rceil) & \blacksquare \end{array}$$

The problem of determining if WS1S sentences are true under the above interpretation is decidable (see, e.g. [Thatcher and Wright, 1967; Thomas, 1990]). The decision procedure is semantically based; it translates a WS1S formula ϕ to an automaton A_ϕ that, essentially, recognizes valuations for the free variables under which ϕ is true. It is easy to determine from the resulting automaton A_ϕ whether a sentence is true in the above structure.

The MONA system implements this decision procedure. Input to MONA is a script consisting of a sequence of definitions followed by a formula ϕ to be proven. MONA computes A_ϕ and, depending on the result, declares ϕ to be valid or delivers a counter-example. MONA is implemented to be as efficient as possible [Henriksen et al, 1995] and works well in practice on a large range of problems despite the non-elementary worst-case complexity of WS1S; empirical evidence of this and an analysis of why this is the case is given in [Basin and Klarlund, 1998].

3 Combining MONA and Isabelle/HOL

We formalize the semantics of WS1S in Isabelle's HOL. We create a theory that formalizes the semantic domains of WS1S (natural numbers and finite sets of natural numbers) as HOL-types. The embedding function (Definition 2) maps WS1S formulae to formulae of this theory and the image of this mapping characterizes what we may call *the WS1S-subset of HOL*. We explain below how we implement the inverse of the embedding so that formulae in this subset of HOL can be submitted to MONA and the results can be incorporated in HOL proofs.

3.1 Logical Basis and Correctness

Hooking an 'oracle' to a theorem prover is risky business. The oracle could be buggy or there could be a mismatch between the semantics of the oracle and semantics of the logic used by the theorem prover. The only way to avoid a buggy oracle is to reconstruct a proof in the theorem prover based on output from the oracle, or perhaps formally verify the oracle itself. For a semantics based decision procedure, proof reconstruction is not a realistic option: one would have to formalize the entire automata-theoretic machinery within HOL and this would amount to verifying the oracle.

By taking a semantic based translation approach (sometimes called semantic embedding in the literature) we cannot avoid the problem of a buggy oracle, but we can formally (although not inside of HOL) show that there is no semantics mismatch and that we do not compromise the consistency of HOL by accepting theorems proved by MONA.

We define in HOL a theory Finset in which we model the semantic domains of WS1S. The set $\mathbb{N}$ is represented by the type nat, which is defined inductively. The Peano axioms can be derived from this definition, including a second-order induction axiom. It follows that every standard model [Andrews, 1986; Gordon and Melham, 1993] of the theory must interpret the type nat (up to isomorphism) as the set $\mathbb{N}$. A similar statement holds for our inductive definition of the type finnat, which is defined as the set of finite subsets of nat. The inductive definition characterizes the finite powerset of $\mathbb{N}$, $\mathcal{F}(\mathbb{N})$, uniquely and every model of the theory must interpret the type finnat (up to isomorphism) as the set $\mathcal{F}(\mathbb{N})$.

Given the above, we can show by induction on the structure of terms and formulae of WS1S that the terms and formulae of WS1S are interpreted in the same way as their image under the embedding function of Definition 2; consequently for every valid WS1S-formula ϕ, the image $\lceil\phi\rceil$ is valid in every standard model of HOL. By soundness of ISABELLE/HOL's deductive system (see [Regensburger, 1994]), we conclude that $\neg\lceil\phi\rceil$ is not derivable in the theory Finset and therefore we may accept $\lceil\phi\rceil$ consistently as a theorem of that theory. To summarize:

THEOREM 1
If an WS1S-formula ϕ is valid in WS1S we can add $\lceil\phi\rceil$ consistently to the theorems of the theory Finset.

3.2 Mechanization

Our goal is to link up two different systems. Mechanically, this entails identifying formulae of an ISABELLE/HOL proof-goal that are in the WS1S-subset of HOL, translating them to WS1S, invoking MONA, and updating the proof when MONA succeeds. These formulae are identified syntactically; they contain only terms of types nat and finnat and operations corresponding to those in WS1S. (see Definition 2). Polymorphic operators such as equality or quantification can be distinguished by their type instance. This allows us to strictly control which formulae are translated.

Typically, users work by adding definitions to theories, thereby extending the language of HOL with new constants, e.g., the constant IF defined by $\mathsf{IF}\, e\, x\, y \equiv (e \to x) \wedge (\neg e \to y)$. When such definitions are also expressible in WS1S, we can include them in our translations. We do this by allowing the user to define corresponding definitions in MONA, and extend the embedding function, provided that the right-hand side of the definition contains only terms that are already in the image of the embedding function. The constants defined in this way are recorded together with their definitions in a particular data structure called a *definition set*. When translating a formula to WS1S, every constant is replaced by a call of the respective macro and the macro definition is added to the declaration part of the generated MONA input-file.

To connect ISABELLE and MONA, we use the general 'oracle interface' provided by Isabelle. Oracles are declared in Isabelle's theory definitions by specifying a top-level ML-function that takes arbitrary data, e.g. a proof goal, and returns a term of Isabelle's meta-level type of truth-values. This function may be an implementation of a decision procedure, a model-checker, or it may call, as in our case, an external reasoning tool.

We have written a tactic that uses the oracle to solve goals arising in HOL proofs. The tactic takes a goal and a definition set, and invokes the oracle. If MONA can prove the resulting formula, then that formula is promoted by the oracle interface to a theorem, which we use to solve the goal; otherwise the tactic fails.

The translation of a subgoal in which one tries to prove a conclusion C from assumptions $A_1, \ldots, A_n$ is organized in several steps. First, we attempt to translate separately each assumption and the conclusion. If this fails for some assumption A_i, then we replace this assumption by true; effectively, we try then to prove the subgoal without A_i. If the translation fails for the conclusion we replace it by false and try to prove a contradiction from the assumptions. This yields the assumptions A'_1, …, A'_n and the conclusion C' in the language of MONA. We join them together as a nested implication from which we take the universal closure and thus obtain a MONA-formula of the form

$$\forall\, X_1 \ldots \forall\, X_k\, \forall x_1 \ldots \forall x_l,:\; A'_1 \to \;\ldots\; \to A'_n \to C'$$

4 Application: Linear Arithmetic

In the previous section we showed how to soundly integrate MONA with HOL by formalizing the semantics of WS1S as a theory-extension of HOL. The resulting coding though is rather 'low level'. It is easier for users to encode problems using higher level concepts like numbers and arithmetic functions as opposed to finite sets and WS1S definable relations on them.

In this section we show how to link these levels in the case of arithmetic. We build a kind of interpreter that allows us to translate arithmetic as expressed in standard Isabelle theories to the (embedded) WS1S theory. This interpretation is done completely within HOL using formally derived rewrite rules; hence correctness is guaranteed.

We will consider several arithmetic theories in this section. For example, *Presburger arithmetic* [Presburger, 1929] is the first-order theory of $(\mathbb{N}, +)$. This theory can easily be extended with inequalities as well as multiplication and division by constants. It also can be extended to a theory of integers by considering pairs of numbers. In [Büchi, 1960] and [Elgot, 1961] Büchi and Elgot note that decidability of WS1S implies that of Presburger arithmetic. After first presenting arithmetic over number representations from finnat, we will consider the first-order theories of $(\mathbb{N}, +)$ and $(\mathbb{Z}, +)$.

4.1 Arithmetic over finnat

We use a binary encoding to represent a natural number n as an element N of type finnat. For all i, $i \in N$ holds if and only if the ith digit in the binary representation of n is a one, e.g., the number $\mathsf{5}$ is represented by the set $\{\!|\, \mathsf{0}, \mathsf{2} \,|\!\}$ since $\mathsf{5} = \mathsf{2}^{\mathsf{0}} + \mathsf{2}^{\mathsf{2}}$. Thus the empty set $\{\!|\,|\!\}$ represents zero and the successor relation is defined as

$$\begin{aligned} \mathsf{SUCC}\, M\, N \equiv\ \ \exists p.\, \forall i.\quad & (i < p \to i \notin M \wedge i \in N) \\ \wedge\quad & (i = p \to i \in M \wedge i \notin N) \\ \wedge\quad & (p < i \to (i \in M) = (i \in N))\,, \end{aligned}$$

i.e., M represents the successor of the number represented by N. To show that this is a proper encoding for the natural numbers, we prove that an analogue of the Peano axioms hold for it; that is we prove the following five formulae.

1) $\{\!|\,|\!\}\ ::\ \mathsf{finnat}$
2) $\forall M.\, \exists N.\, \mathsf{SUCC}\ N\ M$
3) $\forall N.\, \neg\mathsf{SUCC}\ \{\!|\,|\!\}\ M$
4) $\forall K\, L\, M\, N.\, \mathsf{SUCC}\ L\ K\ \wedge\ \mathsf{SUCC}\ N\ M\ \wedge\ L = N\ \to\ K = M$
5) $\forall P.\ (P\, \{\!|\,|\!\})\ \wedge\ (\forall K\, L.\, (P\, K)\ \wedge (\mathsf{SUCC}\, L\, K) \to (P\, L)) \to (\forall N.\, (P\, N))$

The first holds trivially, since $\{\!|\,|\!\}$ is an element of type finnat. We prove the next three automatically using MONA as an oracle. This was a pleasant surprise: our oracle was often useful in proving theorems required to extend its scope. The fifth theorem formalizes an induction principle: all elements of type finnat can be reached from $\{\!|\,|\!\}$ via the successor relation. This formula cannot be proven by MONA since it involves quantification over predicates. However, it can be derived in ISABELLE/HOL using the induction principle associated with the inductively defined finite-set type and in this derivation we can use MONA to discharge different proof obligations.

To use MONA as an arithmetic decision procedure requires defining arithmetic operations for addition, equality, and inequality for numbers encoded as finite sets. The following is an example of encoding addition, i.e. the number represented by S is the sum of the numbers represented by A and B.

$$\begin{aligned} \mathsf{ADD}\, S\, A\, B\ \equiv\ \ & \exists C.\, \mathsf{0} \notin C \wedge \\ & \quad \forall p.\quad \mathsf{mod\_two}\ (p \in A)\ (p \in B)\ (p \in C)\ (p \in S) \\ & \qquad \wedge\ \mathsf{at\_least\_two}\ (p \in A)\ (p \in B)\ (p \in C)\ ((\mathsf{Suc}\, p) \in C) \end{aligned}$$

This implements the standard addition algorithm (over binary representations of numbers) where the pth bit of the result S is the sum of the pth bits of A, B, and the carry (here C) mod 2 and the pth carry is set if at least two of the previous inputs and the carry are set. The auxiliary predicates $\mathsf{mod\_two}$ and $\mathsf{at\_least\_two}$ are defined as follows.

$$\begin{aligned} \mathsf{mod\_two}\, a\, b\, c\, s &\equiv a = b = c = s \\ \mathsf{at\_least\_two}\, a\, b\, c\, d &\equiv d = (a \wedge b) \vee (b \wedge c) \vee (a \wedge c) \end{aligned}$$

4.2 Arithmetic over nat

The above definitions allow us to extend our oracle's definition set and solve arithmetic problems expressed using relations over finnat, i.e., arithmetic problems where numbers are formalized as finite sets representing their binary encodings. This has limited use in practice. Users of HOL usually perform arithmetic reasoning directly within the theory nat. Formally, we can define a translation between these two theories: we recursively define mappings val and bin between the types nat and finnat and prove the following properties, which characterize them as isomorphisms (we have named the properties for subsequent use).

(val_Empty)	$\mathsf{val}\ \{\!\|\!\} = 0$
(val_SUCC)	$\mathsf{SUCC}\ M\ N \implies \mathsf{val}\ M = \mathsf{Suc}(\mathsf{val}\ N)$
(bin_0)	$\mathsf{bin}\ 0 = \{\!\|\!\}$
(bin_Suc)	$\mathsf{bin}\ (\mathsf{Suc}\ n) = (\epsilon M.\ \mathsf{SUCC}\ M\ (\mathsf{bin}\ n))$
$(val\_inverse)$	$\mathsf{val}(\mathsf{bin}\ n) = n$
$(bin\_inverse)$	$\mathsf{bin}(\mathsf{val}\ N) = N$
(ADD)	$\mathsf{ADD}\ (\mathsf{bin}\ s)\ (\mathsf{bin}\ a)\ (\mathsf{bin}\ b) = (s = a + b)$

Next, we automate the use of this encoding, effectively hiding it from the user; we derive rewrite rules as HOL theorems that can be used by Isabelle's simplifier in order to automatically translate arithmetic operations on numbers to formulae built from predicates over sets. Our translation rules are of three kinds. First, rules that replace relations between numbers by relations between sets, e.g.

$$(eq\_c) \qquad (n = m) = (\mathsf{bin}\ n = \mathsf{bin}\ m)$$

Second, rules that propagate applications of bin downwards through the term structure and replace numeric functions by the corresponding predicates over sets. There is a rule for each operation; e.g. for addition we have:

$$(add\_c) \qquad P(\mathsf{bin}(a + b)) = (\exists S.\ \mathsf{ADD}\ S\ (\mathsf{bin}\ a)\ (\mathsf{bin}\ b) \wedge P(S))$$

Third, rules for quantifiers that translate quantification over numbers to quantification over sets, e.g.

$$(all\_c) \qquad (\forall n :: \mathsf{nat}.\ P(n)) = (\forall N :: \mathsf{finnat}.\ P(\mathsf{val}\ N))$$

As previously mentioned, these rules are formally derived in HOL.

A simple example should clarify how the rules together define the translation. Let us consider how rewriting transforms the statement that addition is commutative to a formulae that can be proved by MONA. Rewriting performs the following steps:

$$\forall a\, b.\ a + b = b + a$$

$$\xrightarrow{(eq\_c)} \quad \forall a\, b.\ \mathsf{bin}(a+b) \ = \ \mathsf{bin}(b+a)$$

$$\xrightarrow{(add\_c)} \quad \forall a\, b.\ \exists S.\ \mathsf{ADD}\ S\ (\mathsf{bin}\ a)\ (\mathsf{bin}\ b) \wedge (S = \mathsf{bin}(b+a))$$

$$\xrightarrow{(add\_c)} \quad \begin{array}{l} \forall a\, b.\ \exists S.\ \mathsf{ADD}\ S\ (\mathsf{bin}\ a)\ (\mathsf{bin}\ b) \\ \quad \wedge\ (\ \exists T.\ \mathsf{ADD}\ T\ (\mathsf{bin}\ b)\ (\mathsf{bin}\ a) \wedge (S = T)) \end{array}$$

$$\xrightarrow{(all\_c)}{}^{2} \quad \begin{array}{l} \forall A\, B.\ \exists S.\ \mathsf{ADD}\ S\ (\mathsf{bin}\ (\mathsf{val}\, A))\ (\mathsf{bin}\ (\mathsf{val}\, B)) \\ \quad \wedge\ (\ \exists T.\ \mathsf{ADD}\ T\ (\mathsf{bin}\ (\mathsf{val}\, B))\ (\mathsf{bin}\ (\mathsf{val}\, A)) \wedge (S = T)) \end{array}$$

$$\xrightarrow{(bin\_inverse)}{}^{4} \quad \begin{array}{l} \forall A\, B.\ \exists S.\ \mathsf{ADD}\ S\ A\ B \\ \quad \wedge\ (\ \exists T.\ \mathsf{ADD}\ T\ B\ A) \wedge (S = T)) \end{array}$$

Note that a and b are of type nat whereas A and B are of type finnat. This rewriting is performed by a tactic that is applied before invoking the oracle. Hence the translation is performed automatically and it serves to extend the scope of the oracle to function not just over set based relations like ADD, bin or val, but also to numbers and standard functions and operators on them, which is what users usually require.

4.3 Arithmetic over int

We now turn our attention to the integers and extending our interface to them. Our solution is based on providing a bijection zenc between int and nat. This function maps negative integers to odd natural numbers and positive integers to even natural numbers in the following way:

$$\mathsf{zenc}(z) = \begin{cases} 2 \cdot z & \text{if } \ 0 \leq z \\ -(2 \cdot z + 1) & \text{if } \ z < 0 \end{cases}$$

In our binary representation, this corresponds to shifting the mantissa one digit to the left and using the vacated 0-th bit as sign bit. The inverse of zenc is called zdec. The bijection zbin and it inverse zval between finnat and int are functional compositions of bin with zenc and, respectively, of zdec with val.

$$\mathsf{zbin}(z) \equiv \mathsf{bin}(\mathsf{zenc}(z))$$
$$\mathsf{zval}(Z) \equiv \mathsf{zdec}(\mathsf{val}(Z))$$

Addition for this encoding of the integers can be represented by a predicate ZADD, which is defined similarly to ADD.

Completing the extension is analogous to the extension to nat: We derive rules that characterize zbin as an isomorphism and after that we extend the set of rewriting rules. The extension automatically translates formulae involving integer arithmetic to our encoding of finite sets.

4.4 Experience

Arithmetic reasoning plays an important role in many theorem proving applications such as program verification. Our tactics can be used to automatically discharge first-order arithmetic problems (in the language of 0, +, and inequalities) that arise when using the theories nat and int. It is easy also to extend these procedures too as required. For example, we can add WS1S definable relations like proper subtraction (over the natural numbers), subtraction (over the integers), division or remainder by 2, and the like.

Our tactics operate automatically. Trivial examples like commutativity or associativity of addition are proved in less than two seconds on a Sparc Station-20. As a less trivial example, consider the following verification problem that arises in showing the correctness of our encoding of the integers described in the previous section.

$$\mathsf{ZADD}\ s\ a\ b = (\mathsf{zval}\ s = \mathsf{zval}\ a + \mathsf{zval}\ b)$$

Expanding the definitions of zval and zdec yields the following goal.

$$
\begin{array}{l}
\quad(\mathsf{val}\ s\ \mathsf{mod}\ 2 = 0 \rightarrow \\
\qquad\quad(\mathsf{val}\ a\ \mathsf{mod}\ 2 = 0 \rightarrow \\
\qquad\qquad\quad(\mathsf{val}\ b\ \mathsf{mod}\ 2 = 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = (\mathsf{val}\ s\ \mathsf{div}\ 2 = \mathsf{val}\ a\ \mathsf{div}\ 2 + \mathsf{val}\ b\ \mathsf{div}\ 2)) \\
\qquad\qquad\wedge\ (\mathsf{val}\ b\ \mathsf{mod}\ 2 \neq 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = (\mathsf{Suc}(\mathsf{val}\ s\ \mathsf{div}\ 2 + \mathsf{val}\ b\ \mathsf{div}\ 2) = \mathsf{val}\ a\ \mathsf{div}\ 2))) \\
\qquad\wedge\ (\mathsf{val}\ a\ \mathsf{mod}\ 2 \neq 0 \rightarrow \\
\qquad\qquad\quad(\mathsf{val}\ b\ \mathsf{mod}\ 2 = 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = (\mathsf{Suc}(\mathsf{val}\ s\ \mathsf{div}\ 2 + \mathsf{val}\ a\ \mathsf{div}\ 2) = \mathsf{val}\ b\ \mathsf{div}\ 2)) \\
\qquad\qquad\wedge\ (\mathsf{val}\ b\ \mathsf{mod}\ 2 \neq 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = \mathsf{False}))) \\
\wedge\ (\mathsf{val}\ s\ \mathsf{mod}\ 2 \neq 0 \rightarrow \\
\qquad\quad(\mathsf{val}\ a\ \mathsf{mod}\ 2 = 0 \rightarrow \\
\qquad\qquad\quad(\mathsf{val}\ b\ \mathsf{mod}\ 2 = 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = \mathsf{False}) \\
\qquad\qquad\wedge\ (\mathsf{val}\ b\ \mathsf{mod}\ 2 \neq 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = (\mathsf{val}\ b\ \mathsf{div}\ 2 = \mathsf{val}\ a\ \mathsf{div}\ 2 + \mathsf{val}\ s\ \mathsf{div}\ 2))) \\
\qquad\wedge\ (\mathsf{val}\ a\ \mathsf{mod}\ 2 \neq 0 \rightarrow \\
\qquad\qquad\quad(\mathsf{val}\ b\ \mathsf{mod}\ 2 = 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = (\mathsf{val}\ a\ \mathsf{div}\ 2 = \mathsf{val}\ b\ \mathsf{div}\ 2 + \mathsf{val}\ s\ \mathsf{div}\ 2)) \\
\qquad\qquad\wedge\ (\mathsf{val}\ b\ \mathsf{mod}\ 2 \neq 0 \rightarrow \mathsf{ZADD}\ s\ a\ b = (\mathsf{Suc}(\mathsf{val}\ a\ \mathsf{div}\ 2 + \mathsf{val}\ b\ \mathsf{div}\ 2) = \mathsf{val}\ s\ \mathsf{div}\ 2))))
\end{array}
$$

Such a goal would be rather tedious for a human to prove interactively. It is solved by our tactic in 43.6 seconds. Interestingly though (and a possible area for improvement), almost all the time is spent on rewriting; MONA requires only 0.13 seconds to verify the translated formula.

5 Application: Circuit Verification

To illustrate the generality and flexibility of our combination, we present a second example from a rather different domain: formal reasoning about parametric circuit descriptions. As previously noted, systems with multiple independent parameters fall out of the scope of WS1S but can be formalized in HOL. Our verification strategy is to reduce the number of parameters to a single parameter, at which point the result may be WS1S formalizable.

Here we consider an example of a circuit with two parameters: time, and bit-width. The key idea is that we can eliminate the time parameter (eliminating bit-width is more difficult, but also possible) by reducing the correctness problem to showing that an invariant holds over consecutive time-instances. We demonstrate this with the verification of a parameterized family of n-bit counters. Due to space limitations we only sketch the main ideas; full formalization and proof details are given in [Friedrich, 1998].

In hardware verification, like software verification, one establishes that an implementation fulfills its specification. A common approach is to formalize both the implementation and the specification as relations between the circuit's inputs and outputs [Camilleri *et al.*, 1986]. Circuits are built from primitive relations corresponding to transistors, gates, and the like, are combined by conjunction, and 'wired together' using shared variables, hidden by existential quantifiers. For sequential circuits, signals are modelled as functions from time (discrete in our case) to port-values, which are truth values. It is often desirable to establish the correctness of parametric families of circuits. That is, rather than establish that an n-bit counter is correct for particular values of n, we show that the entire family is correct for all $n \in \mathsf{nat}$. Note that parametric circuits require parametric signals (busses); in our work we model these as functions from time to elements of type finnat with the convention that the i-th bit of a bus B is set at time t if and only if $i \in B(t)$.

Figure 1 contains a diagram of a schematic n-bit counter. It takes an n-bit bus IN as input and yields an n-bit output OUT and a carry-out bit co. In addition, it takes other input signals, namely pe (parallel enable) and ce (count enable), which control the function performed by the counter. The components INC, MUX and DLATCH are parameterized implementations of an n-bit incrementer, multiplexor and dlatch, respectively. Assuming that these have been modelled in HOL, we model the implementation of the n-bit counter by the following formulae.

$$\begin{aligned}
&\mathsf{COUNTER}\ n\ IN\ pe\ ce\ OUT\ co\ \equiv \\
&\qquad \exists\, D\ W\ u\ v.\ \mathsf{INC}\ n\ OUT\ ce\ W\ u\ \wedge\ \mathsf{MUX}\ n\ pe\ IN\ W\ D \\
&\qquad\qquad \wedge\ \mathsf{DLATCH}\ n\ D\ OUT\ \wedge\ \mathsf{NOT}\ pe\ v\ \wedge\ \mathsf{AND}\ v\ u\ co
\end{aligned}$$

This circuit is explicitly parameterized by the bit-width n. The time pa-

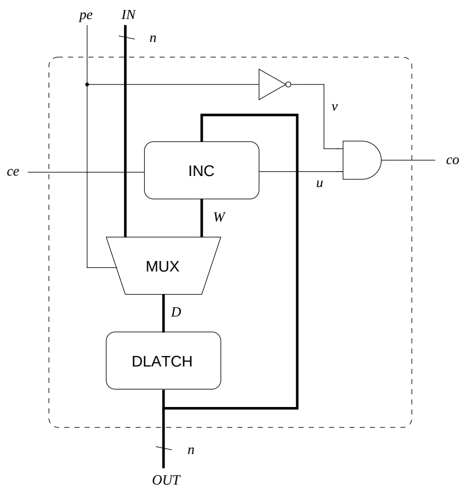

Figure 1: Implementation of the n-bit counter

rameter is implicitly part of the specification as all inputs are signals, which are functions of time.

The values of pe and ce determine the counter's action. For any time instance t, if pe holds at time t (written by applying the function pe to the argument t) then, independently of ce, the value output by the counter at time $t+1$ is that of the IN-bus at time t. Otherwise, if ce is set, the value output at time $t+1$ is one greater than that output at time t, unless there is an overflow, in which case the output value is 0. Finally, if neither pe nor ce is set at time t, the output does not change from time t to time $t+1$. The following HOL formula makes this precise.

```
$\mathsf{COUNTER\_SPEC}\ n\ t\ IN\ pe\ ce\ OUT\ co \equiv$
    if $pe\ t$
    then $\mathsf{valn}\ n\ (OUT(\mathsf{Suc}\ t)) = \mathsf{valn}\ n\ (IN\ t)\ \wedge\ \neg\ co\ t$
    else if $ce\ t$
        then if $\mathsf{MAX}\ n\ (OUT\ t)$
            then $\mathsf{valn}\ n\ (OUT(\mathsf{Suc}\ t)) = 0\ \wedge\ co\ t$
            else $\mathsf{valn}\ n\ (OUT(\mathsf{Suc}\ t)) = \mathsf{Suc}(\mathsf{valn}\ n(OUT\ t))\ \wedge\ \neg\ co\ t$
        else $\mathsf{valn}\ n\ (OUT(\mathsf{Suc}\ t)) = \mathsf{valn}\ n\ (OUT\ t)\ \wedge\ \neg\ co\ t$
```

We can now state what it means for the counter to be correctly implemented as an equivalence between the implementation and the specification.

$$\mathsf{COUNTER}\ n\ IN\ pe\ ce\ OUT\ co\ =\ \forall t.\ \mathsf{COUNTER\_SPEC}\ n\ t\ IN\ pe\ ce\ OUT\ co$$

All free variables are implicitly universally quantified, i.e., we show the equivalence for all possible port-values. As previously noted, the time parameter on the left-hand side is implicit in our use of signals. This parameter is explicit however, in the specifications of the components $\mathsf{INC\_SPEC}$, $\mathsf{MUX\_SPEC}$, etc.

The counter is proved correct in three steps. First we expand its definition and replace the implementations of the components by their specifications (these subcomponents are first proved correct with respect to their specifications). This reveals the implicit time parameter on the left-hand side and results in the following goal:

$$\begin{aligned}
(\exists\ D\ \ W\ \ u\ v.\ & (\forall t.\ \mathsf{INC\_SPEC}\ t\ n\ \mathit{OUT}\ \mathit{ce}\ \ W\ \ u) \\
& \wedge (\forall t.\ \mathsf{MUX\_SPEC}\ t\ n\ \mathit{pe}\ \mathit{IN}\ \ W\ \ D) \\
& \wedge (\forall t.\ \mathsf{DLATCH\_SPEC}\ t\ n\ D\ \mathit{OUT}) \\
& \wedge (\forall t.\ \mathsf{NOT\_SPEC}\ t\ \mathit{pe}\ \ v) \\
& \wedge (\forall t.\ \mathsf{AND\_SPEC}\ t\ v\ \ u\ \ \mathit{co})) \\
= (\forall t.\ \mathsf{COUNTER\_SPEC}\ & n\ t\ \mathit{IN}\ \mathit{pe}\ \mathit{ce}\ \mathit{OUT}\ \ \mathit{co})
\end{aligned}$$

In the second step we eliminate the time parameter. To do this we pull the universal quantifier outside the equivalence using derived inference rules for quantification. Pulling the universal quantifier over an existential quantifier can be seen as a kind of reverse Skolemization where the (internal) signals D and W, which are functions of type $\mathsf{nat} \Rightarrow \mathsf{finnat}$, are replaced by port values D' and W' of type finnat and where the signals u and v, which are of type $\mathsf{nat} \Rightarrow \mathsf{bool}$, are replaced by port values of type bool, respectively. Moreover, pulling the time quantifier over the equivalence corresponds to choosing a fixed (but arbitrary) time point at which implementation and specification are compared. This step results in the following proof obligation:

$$\begin{aligned}
\forall t.(\exists\ D'\ \ W'\ \ u'\ \ v'.\ & \mathsf{INC\_SPEC}'\ t\ n\ \mathit{OUT}\ \ \mathit{ce}\ \ W'\ \ u' \\
& \wedge \mathsf{MUX\_SPEC}'\ t\ n\ \mathit{pe}\ \mathit{IN}\ \ W'\ D' \\
& \wedge \mathsf{DLATCH\_SPEC}'\ t\ n\ D'\ \mathit{OUT} \\
& \wedge \mathsf{NOT\_SPEC}'\ t\ \mathit{pe}\ \ v' \\
& \wedge \mathsf{AND\_SPEC}'\ t\ v'\ \ u'\ \ \mathit{co}) \\
= (\mathsf{COUNTER\_SPEC}\ & n\ t\ \mathit{IN}\ \ \mathit{pe}\ \mathit{ce}\ \mathit{OUT}\ \ \mathit{co})
\end{aligned}$$

After this, we can simplify matters a bit by removing universal quantifiers and replacing all remaining applications of signals to time points by fresh variables that represent the values of the signals at these time points. For example, we replace all occurrences of $\mathit{OUT}\ t$ with the variable $\mathit{OUT0}$ and all occurrences of $\mathit{OUT}(\mathsf{Suc}\ t)$ with the variable $\mathit{OUT1}$.

This leaves us with a new proof goal that no longer contains the time parameter t. (We have here expanded the definitions of $\mathsf{INC\_SPEC}'$, $\mathsf{MUX\_SPEC}'$ etc.)

$$
\begin{array}{l}
\quad(\exists\, D'\ W'\ u'\ v'.(\ \ \mathsf{if}\ ce0 \\
\qquad\qquad\qquad\qquad\ \mathsf{then}\ \ \mathsf{if}\ \mathsf{MAX}\ n\ OUT0 \\
\qquad\qquad\qquad\qquad\qquad\quad\ \mathsf{then}\ \mathsf{valn}\ n\ W' = 0\ \wedge\ u' \\
\qquad\qquad\qquad\qquad\qquad\quad\ \mathsf{else}\ \mathsf{valn}\ n\ W' = \mathsf{Suc}(\mathsf{valn}\ n\ OUT0)\ \wedge\ \neg u' \\
\qquad\qquad\qquad\qquad\ \mathsf{else}\ \mathsf{valn}\ n\ W' = \mathsf{valn}\ n\ OUT0\ \wedge\ \neg u') \\
\qquad\qquad\qquad\quad\ \wedge\ (\mathsf{if}\ pe0\ \mathsf{then}\ \mathsf{EQ}\ n\ D'\ IN0\ \mathsf{else}\ \mathsf{EQ}\ n\ D'\ W') \\
\qquad\qquad\qquad\quad\ \wedge\ \mathsf{EQ}\ n\ OUT1\ D' \\
\qquad\qquad\qquad\quad\ \wedge\ v' = (\neg pe0) \\
\qquad\qquad\qquad\quad\ \wedge\ co0 = (v'\ \wedge\ u')) \\
= (\ \ \mathsf{if}\ pe0 \\
\qquad \mathsf{then}\ \mathsf{valn}\ n\ OUT1 = \mathsf{valn}\ n\ IN0\ \wedge\ \neg co0 \\
\qquad \mathsf{else}\ \ \mathsf{if}\ ce0 \\
\qquad\qquad\quad \mathsf{then}\ \ \mathsf{if}\ \mathsf{MAX}\ n\ OUT0 \\
\qquad\qquad\qquad\qquad \mathsf{then}\ \mathsf{valn}\ n\ OUT1 = 0\ \wedge\ co0 \\
\qquad\qquad\qquad\qquad \mathsf{else}\ \mathsf{valn}\ n\ OUT1 = \mathsf{Suc}(\mathsf{valn}\ n\ OUT0)\ \wedge\ \neg co0 \\
\qquad\qquad\quad \mathsf{else}\ \mathsf{valn}\ n\ OUT1 = \mathsf{valn}\ n\ OUT0\ \wedge\ \neg co0)
\end{array}
$$

This is WS1S formalizable. Invoking our oracle is the third step and this completes the proof in 20 seconds.

6 Related Work and Conclusions

6.1 Related Work

The combination of logics and provers based on them is an active area of research. Various groups have investigated decision procedures for arithmetic reasoning and their integration with theorem provers. The theory of linear integer arithmetic was shown to be decidable by Presburger in 1929. The application of WS1S to arithmetic was documented by Büchi in [Büchi, 1960]. Boyer and Moore have integrated a procedure for the universal fragment of linear natural arithmetic in their prover NQTHM [Boyer and Moore, 1988]. The PVS system, based on higher-order logic, implements a solver for linear equations [Owre *et al.*, 1992], and the STEP system combines decision procedures for partial orders and linear integer arithmetic with a semi-decision procedure for first-order logic [Manna *et al.*, 1994].

In each of the above combinations, the decision procedure is part of the prover and is 'hardwired' to operate over specific data-types and syntactic formulae classes. Such close coupling avoids translation and linkage to an external prover, but it introduces inflexibility. As documented by Boyer and Moore [Boyer and Moore, 1988], in practice few subgoals fit the requirements of such decision procedures; their successful use requires closely integrating the procedure with other theorem proving activities. Our formalization, which is based on an open coupling between WS1S and HOL, offers advantages in this regard. WS1S is a decision procedure for a general theory of finite sets of natural numbers in which arithmetic can be encoded. The encoding is made by, and transparent to, the user and can easily be modified to different HOL datatypes (e.g., natural numbers or integers). Additional relations in HOL can be user defined and incorporated into the translation, provided they are WS1S expressible.

Relevant to our second application is research on using model checkers to reason about systems with infinite or parameterized state spaces, which cannot directly be model-checked. Kurshan and Lamport [Kurshan and Lamport, 1993] have presented a similar connection between a proof checker for TLA and the COSPAN model-checker; they present a parameterized n-bit multiplier verified in TLA based on induction and the verification of an 8-bit multiplier by COSPAN. The interaction between the two systems is very loose: both systems were used independently leaving the users to communicate between them and to insure correctness of the interaction.

Our parameter elimination technique is closely related to work of Wolper and Lovinfosse [Wolper and Lovinfosse, 1989] and Kurshan and McMillan [Kurshan and McMillan, 1989]. They present techniques where model checking is used to verify an invariant that formalizes an induction step. Our work differs from theirs in that we use a general purpose decision procedure (as opposed to a model checker) and the parameter elimination steps are completely formalized within HOL. This kind of formalized reduction is also found in work at SRI where a decision procedure for the mu-calculus is implemented in PVS and used to reason about infinite state systems, which can be reduced to finite state systems using abstraction techniques [Rajan *et al.*, 1995] (see also [Müller and Nipkow, 1995]).

6.2 Conclusions

Our experience with the combination is positive. WS1S has a simple semantics that leads to a simple embedding whose correctness is easily established. Despite this simplicity, WS1S is expressive, has a wide range of application, and one can easily embed or combine other decision procedures with WS1S. The rather different nature of our two examples is evidence of this flexibility.

Our work is just a starting point, and there are many interesting open problems. Complexity is one of them. WS1S is non-elementary in worst case; however, MONA can solve many complex problems quickly in practice. Theoretical reasons for this and elementary bounds for certain sub-problems are given in [Basin and Klarlund, 1998]. Theoretical and practical comparisons with other decision procedures for arithmetic constitute future work. Another question concerns the scope and applicability of our embedding. In the case of arithmetic, we can precisely state when our decision procedure is applicable. For other applications, like verification of parameterized sequential systems, the situation is not so clear: Although a reduction from an undecidable class of problems to a decidable one cannot always succeed, perhaps there are useful characterizations of when it can work.

References

[Andrews, 1986] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, 1986.

[Basin and Klarlund, 1998] David Basin and Nils Klarlund. Automata based symbolic reasoning in hardware verification. *The Journal of Formal Methods in Systems Design*, 1998. To appear.

[Boyer and Moore, 1988] R. S. Boyer and J. S. Moore. Integrating decision procedures into heuristic theorem provers: A case study with linear arithmetic. *Machine Intelligence*, (11):83–124, 1988.

[Büchi, 1960] J. R. Büchi. Weak second order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[Camilleri *et al.*, 1986] A. J. Camilleri, M. J. C. Gordon, and T. F. Melham. Hardware verification using higher-order logic. In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs.* North Holland, September 1986.

[Elgot, 1961] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98:21–52, 1961.

[Friedrich, 1998] Stefan Friedrich. Integration of a Decision Procedure for Second-Order Monadic Logic in a Higher-Order Logic Theorem Proving Environment. Master's thesis, Universität des Saarlandes, 1998.

[Gordon and Melham, 1993] Mike J. C. Gordon and Tom F. Melham. *Introduction to HOL.* Cambridge University Press, 1993.

[Henriksen et al, 1995] Jesper G. Henriksen et al. Mona: Monadic second-order logic in practice. In Ed Brinksma et al, editor, *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS'95*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110, Heidelberg, May 1995. Springer-Verlag.

[Kurshan and Lamport, 1993] Robert Kurshan and Leslie Lamport. Verification of a multiplier: 64 bits and beyond. In Costas Courcoubetis, editor, *Proceedings of the Conference on Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 166–179, Heidelberg, 1993. Springer-Verlag.

[Kurshan and McMillan, 1989] Robert Kurshan and Ken McMillan. A structural induction theorem for processes. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*, pages 239–247. ACM Press, 1989.

[Manna *et al.*, 1994] Zohar Manna, Anuchit Anuchitanukul, Nikolaj Bjorner, Anca Browne, Edward Chang, Michael Colon, Luca de Alfaro, Harish Devarajan, Henny Sipma, and Tomas Uribe. STeP: The stanford temporal prover. Technical Report CS-TR-94-1518, Stanford University, Computer Science Department, June 1994.

[Müller and Nipkow, 1995] Olaf Müller and Tobias Nipkow. Combining model checking and deduction for I/O-automata. In Ed Brinksma et al, editor, *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS'95*, volume 1019 of *Lecture Notes in Computer Science*, pages 1–16, Heidelberg, May 1995. Springer-Verlag.

[Owre *et al.*, 1992] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proc. of the 11th Intern. Conf. on Autom. Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Heidelberg, 1992. Springer-Verlag.

[Paulson, 1994] Lawrence C. Paulson. *Isabelle : a generic theorem prover; with contributions by Tobias Nipkow*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 1994.

[Presburger, 1929] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchen, die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congrès des Mathématicienes des Pays Slaves*, pages 92–101, 395, Warsaw, 1929.

[Rajan *et al.*, 1995] S. Rajan, N. Shankar, and M.K. Srivas. An integration of model-checking with automated proof checking. In Pierre Wolper, editor, *Computer-Aided Verification, CAV '95*, volume 939 of *Lecture Notes in Computer Science*, pages 84–97, Heidelberg, June 1995. Springer-Verlag.

[Regensburger, 1994] Franz Regensburger. *HOLCF: Eine konservative Erweiterung von HOL um LCF.* PhD thesis, Technische Universität München, November 1994.

[Thatcher and Wright, 1967] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, August 1967.

[Thomas, 1990] Wolfgang Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4. MIT Press/Elsevier, 1990.

[Wolper and Lovinfosse, 1989] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 68–80, Heidelberg, June 1989. Springer-Verlag.