Model Checking the First-Order Fragment of Higher-Order Fixpoint Logic

Roland Axelsson[1] and Martin Lange[2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
[2] Department of Computer Science, University of Aarhus, Denmark

Abstract. We present a model checking algorithm for HFL1, the first-order fragment of Higher-Order Fixpoint Logic. This logic is capable of expressing many interesting properties which are not regular and, hence, not expressible in the modal μ-calculus. The algorithm avoids best-case exponential behaviour by localising the computation of functions and can be implemented symbolically using BDDs.

We show how insight into the behaviour of this procedure, when run on a fixed formula, can be used to obtain specialised algorithms for particular problems. This yields, for example, the competitive antichain algorithm for NFA universality but also a new algorithm for a string matching problem.

1 Introduction

Properties (of words or trees) that can be expressed in the modal μ-calculus are at most regular, i.e. they can also be defined by a finite automaton [4]. The expressive power of temporal logics like LTL, CTL etc. is even strictly below that of full ω-regularity. Nevertheless, there are many natural examples of interesting properties that are not regular in the language theoretic sense, for example: detection of buffer underflows, unlimited counting, repetition of sequences of actions, etc.

Non-regular properties have recently attracted more and more attention, and the need for algorithmic handling of such properties is about to become accepted. This is for example manifested in CaRet, a specification formalism for linear time properties [1]. It can express some, but not all context-free languages.

Here we use the (branching temporal) fixpoint logic HFL1, the first-order fragment of *Higher-Order Fixpoint Logic* (HFL) [9]. HFL achieves high expressive power by combining the modal μ-calculus with a simply typed λ-calculus. We show that the first-order fragment which is obtained by restricting the λ-calculus part to first-order functions, is already capable of expressing interesting program properties like the ones mentioned above. Moreover, it turns out that many other problems – not necessarily from program verification only – can be reduced to the model checking problem for HFL1, e.g. the universality problem for non-deterministic finite automata (NFA-UNIV), the evaluation problem for quantified Boolean formulas (QBF), the satisfiability problem for modal logic

(K-SAT), etc. Commonly, these problems can be *expressed* in HFL1 in the sense that there is a *fixed* formula whose models are exactly the (encodings of) positive instances to such problems. Note that HFL1's model checking problem is EXPTIME-complete [2]. This comparably high computational complexity is necessarily the price to pay for the increased expressive power.

It is straight-forward to extend a symbolic model checking algorithm for $\mathcal{L}_\mu$ to HFL1 by doing fixpoint iterations in function space lattices. This, however, yields a *best-case* running time that is exponential in the size of the underlying model. Sect. 3 presents a model checking algorithm for HFL1 that localises the computation of fixpoints in function spaces and, hence, shows exponential behaviour in the *worst case* only.

Our algorithm exploits the fact that in order to model check an HFL1 property, one usually does not need to know the value of a function on all elements of its domain. Instead it computes these functions in a demand-driven fashion. Due to fixpoint operators the value of a function may be defined recursively, i.e. it may depend on the value of the same function on a different argument. By incorporating into fixpoint iteration the collecting of such function arguments we approximate the total functions in the HFL1 semantics by partial ones that agree with them on those arguments that cause the demand.

In static program analysis this technique is known as *neededness analysis* [5].[1] It resembles *local model checking* – avoiding computations that are unnecessary for the verification task. On the other hand, the algorithm is *global* in the sense that it computes in one go all states of a transition system that satisfy the given formula.

The algorithm works on sets of states using only standard operations. Thus, it can be implemented fully symbolically, i.e. using BDDs to represent state sets and transitions.

Below we first present HFL1 and give a few examples of expressible properties that are of interest in program verification. This is followed by the model checking algorithm and the proof of its soundness and completeness. The rest of the paper focuses on the "model checking is more than program verification" aspect mentioned above. We exemplarily pick out NFA-UNIV and show how it can be expressed in HFL1. We then use the Tabakov/Vardi model of random NFAs [8] to measure the gain of local fixpoint computations in comparison to the naïve extension of the modal μ-calculus model checker which would do fixpoint iterations in the function space lattice. We then show how to take advantage of the fact that NFA-UNIV can be reduced to the model checking problem for HFL1 on a *fixed formula*. This induces a special instance of our model checking algorithm which can be optimised w.r.t. that fixed formula. It turns out that this instance coincides with the competitive algorithm for NFA-UNIV by Henzinger et al., based on antichains [10]. We conclude by discussing further extensions of the model checking algorithm and further special instances for other problems like the ones mentioned above.

[1] Note that this has nothing to do with lazy evaluation in functional programming, let alone non-strict evaluation of higher-order functions!

2 The First-Order Fragment of HFL

Let Σ be a finite set of action names, $\mathcal{P}$ be an at most countably infinite set of propositions and $\mathcal{V}$ a countably infinite set of variable names. Formulas of HFL1 in positive normal form[2] are given by the following grammar.

$$\begin{aligned}\varphi &:= q \mid \neg q \mid X \mid \neg X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \varphi\ \varphi \mid \lambda X.\varphi \mid \mu X^\tau.\varphi \mid \nu X^\tau.\varphi \\ \tau &:= \mathrm{Pr} \mid \mathrm{Pr} \rightarrow \tau\end{aligned}$$

where $a \in \Sigma$, $q \in \mathcal{P}$, and $X \in \mathcal{V}$. We require each fixpoint variable X to only occur positively in its binding formula $\sigma X.\varphi$, for some $\sigma \in \{\mu, \nu\}$, and never to be bound more than once. Hence, each variable comes with a unique fixpoint type μ or ν.

The τ are called HFL1-types. Note that each one is of the form $\tau_k = \mathrm{Pr} \rightarrow \ldots \rightarrow \mathrm{Pr} \rightarrow \mathrm{Pr}$ with k function arrows in it. The type τ_0 is used to model *predicates*, and τ_k for $k \geq 1$ models k-ary *predicate transformers*. A predicate is the same as a 0-ary predicate transformer.

Type annotations are used in order to avoid polymorphic effects. With these annotations, each formula obtains a unique type. We assume formulas to be well-typed using standard typing rules from the simply typed λ-calculus. For example, q, $\langle a \rangle\varphi$, $\varphi_1 \vee \varphi_2$ all have type Pr. In $\varphi\ \psi$, formula ψ must have type Pr, and φ must have type τ_k for some $k \geq 1$. The type of $\varphi = \lambda X.\psi$ can be inferred assuming that X has type Pr: if ψ has type τ_k then φ has type τ_{k+1}, etc.

Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ be a transition system with state set $\mathcal{S}$, binary transition relations $\xrightarrow{a}$ for every $a \in \Sigma$, and a labeling function $L : \mathcal{P} \rightarrow 2^{\mathcal{S}}$ that assigns to every atomic proposition the set of states in which it is true.

We write $\mathcal{D}(\mathcal{S})$ for the domain of k-ary predicate transformers, $k \geq 0$. Formally, $\mathcal{D}(\mathcal{S})$ is the least fixpoint of the domain equation $\mathcal{X} = 2^{\mathcal{S}} + (2^{\mathcal{S}} \rightarrow \mathcal{X})$. Note that $\mathcal{D}(\mathcal{S}) \simeq \bigcup_{k \in \mathbb{N}} \mathcal{D}^k(\mathcal{S})$ where $\mathcal{D}^k(\mathcal{S}) = 2^{\mathcal{S}} \rightarrow \ldots \rightarrow 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$ with k arrows in it. Each $\mathcal{D}^k(\mathcal{S})$ forms a complete lattice with pointwise inclusion ordering $\sqsubseteq$ and meets $\sqcap$ and joins $\sqcup$. In the case of $k = 0$, these operations simply boil down to $\subseteq$, $\cap$ and $\cup$.

Let $\rho : \mathcal{V} \rightarrow \mathcal{D}(\mathcal{S})$ be an environment mapping variables to predicate transformers. The semantics of HFL1 is explained as follows. Note that a subformula of the form $\neg X$ can only be of type Pr, and X must be λ-bound in this case.

$$\begin{aligned}
[\![q]\!]^{\mathcal{T}}_\rho &:= L(q) \\
[\![\neg q]\!]^{\mathcal{T}}_\rho &:= \mathcal{S} \setminus L(q) \\
[\![X]\!]^{\mathcal{T}}_\rho &:= \rho(X) \\
[\![\neg X]\!]^{\mathcal{T}}_\rho &:= \mathcal{S} \setminus \rho(X) \\
[\![\varphi \vee \psi]\!]^{\mathcal{T}}_\rho &:= [\![\varphi]\!]^{\mathcal{T}}_\rho \cup [\![\psi]\!]^{\mathcal{T}}_\rho \\
[\![\varphi \wedge \psi]\!]^{\mathcal{T}}_\rho &:= [\![\varphi]\!]^{\mathcal{T}}_\rho \cap [\![\psi]\!]^{\mathcal{T}}_\rho
\end{aligned}$$

[2] Positive normal form does not impede the expressive power but simplifies the presentation of the semantics.

$$\begin{array}{rcl}
[\![\langle a\rangle\varphi]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid \exists t \in [\![\varphi]\!]^{\mathcal{T}}_{\rho} \text{ s.t. } s \xrightarrow{a} t\} \\
[\![[a]\varphi]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid \forall t \in \mathcal{S} : s \xrightarrow{a} t \text{ implies } t \in [\![\varphi]\!]^{\mathcal{T}}_{\rho}\} \\
[\![\varphi\ \psi]\!]^{\mathcal{T}}_{\rho} & := & [\![\varphi]\!]^{\mathcal{T}}_{\rho}([\![\psi]\!]^{\mathcal{T}}_{\rho}) \\
[\![\lambda X.\varphi]\!]^{\mathcal{T}}_{\rho} & := & \lambda T.[\![\varphi]\!]^{\mathcal{T}}_{\rho[X\mapsto T]} \text{ for } T \in 2^{\mathcal{S}} \\
[\![\mu X^{\tau_k}.\varphi]\!]^{\mathcal{T}}_{\rho} & := & \bigsqcap \{f \in \mathcal{D}^k \mid [\![\varphi]\!]^{\mathcal{T}}_{\rho[X\mapsto f]} \sqsubseteq f\} \\
[\![\nu X^{\tau_k}.\varphi]\!]^{\mathcal{T}}_{\rho} & := & \bigsqcup \{f \in \mathcal{D}^k \mid f \sqsubseteq [\![\varphi]\!]^{\mathcal{T}}_{\rho[X\mapsto f]}\}
\end{array}$$

The modal μ-calculus is easily seen to be a fragment of HFL1 and is in fact obtained by disallowing λ-abstraction and function application which implies that all subformulas are predicates only. HFL1 also subsumes FLC [9] which is basically obtained by restricting all types to τ_0 and τ_1 only.

Example 1. The following formula is true in a model iff it is bisimilar to a balanced tree. Note that bisimulation-invariance is an inherent property of HFL1, because it can be embedded into infinitary modal logic which in turn is incapable of distinguishing bisimilar models. It is therefore not possible to state that the model indeed *is* a balanced tree.

$$\varphi_{bal} \;:=\; \big(\nu X^{\tau_2}.\lambda Z.\lambda Y.(\neg Z \vee \neg Y) \wedge (X\ \langle -\rangle Z\ \langle -\rangle Y)\big)\ \langle -\rangle \mathtt{tt}\ [-]\mathtt{ff}$$

This is best understood by unfolding the fixpoint formula to an infinite conjunction. It then simply says: for all $k \in \mathbb{N}$ it is not the case that both $\langle -\rangle^{k+1}\mathtt{tt}$ and $\langle -\rangle^{k}[-]\mathtt{ff}$ hold, i.e. if there is a path of length at least $k+1$ then there is no maximal path of length k only.

This property can be used to show for example that all runs of a non-deterministic program terminate after the same number of steps. It is also closely related to Emerson's *uniform inevitability* [3].

Example 2. HFL1 can easily express the absence of underflows in unbounded buffers – due to unboundedness clearly not a regular property.

$$\varphi_{buf} \;:=\; \mu X^{\tau_1}.(\lambda Z.\langle \mathtt{out}\rangle Z \vee \langle \mathtt{in}\rangle(X\ (X\ Z)))\ \mathtt{tt}$$

This formula is best understood by comparing it to the CFG $X \to \mathtt{out} \mid \mathtt{in}\,X\,X$. It generates the language of all words $w = u\mathtt{out}$ s.t. $|u|_{\mathtt{in}} = |u|_{\mathtt{out}}$ and for all prefixes v of w we have: $|v|_{\mathtt{in}} \geq |v|_{\mathtt{out}}$. These are exactly the prefixes of buffer runs which are violating due to an underflow.

Of course, for buffers of fixed capacity n this property can easily be expressed in the modal μ-calculus. Being fixed is not only sufficient but also necessary for definability in the μ-calculus. Hence, there are two distinct advantages that logics like HFL1 have over the μ-calculus.

– HFL1 enables *black-box* verification when the exact size of the underlying finite model is unknown.
– Properties like the ones above can be formalised in HFL1 using a *fixed* formula, whereas expressing them in the μ-calculus requires a family of formulas that grow at least linearly in the size of the models.

3 Model Checking HFL1

A naïve extension of the standard global model checker for the modal μ-calculus would represent the semantics of a subformula of type τ_k as a table of the following form. Let $\mathcal{S} = \{s_0, s_1, \dots, s_{n-1}\}$ be the finite state space of the underlying transition system.

arg_1	$\emptyset$	$\{s_0\}$	$\{s_1\}$	$\dots$	$\{s_0, s_1\}$	$\dots$	$\mathcal{S}$	$\emptyset$	$\{s_0\}$	$\dots$
arg_2	$\emptyset$	$\emptyset$	$\emptyset$	$\dots$	$\emptyset$	$\dots$	$\emptyset$	$\{s_0\}$	$\{s_0\}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
arg_k	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\dots$	$\dots$	$\emptyset$	$\emptyset$	$\emptyset$	$\dots$
$value$	T_0	T_1	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

It simply lists the value of that function for every possible argument from its finite domain. Note that this has uncurried the function silently. Fixpoint iteration can be done on these objects in the same way. For a least fixpoint such a table is initialised with $T_i = \emptyset$ for all i. Successive values of this function approximating the least fixpoint can be found by iterating the corresponding functional on these values and updating the table. Note that the width of the table is 2^{kn}. Hence, a fixpoint iteration might take $O(n \cdot 2^{kn})$ many steps due to monotonicity. This leads, for any $\varphi \in$ HFL1, to an exponential model checking algorithm for this fixed φ.

However, this procedure can be localised. It is never necessary to know the entire value of a function. Functions only occur as applicators, i.e. only their value on certain arguments are needed for the model checking problem. Only in the worst case and when embedded in a fixpoint iteration, the value of a function on all possible arguments might be required. This leads to the following idea of a localised model checker for HFL1, depicted in Fig. 1.

Each variable X of type τ_k in the input formula φ_0 is associated with a *partial* function of type $\mathcal{D}^k(\mathcal{S}) \to 2^{\mathcal{S}}$ represented for example by a table as shown above. Since they are allowed to be partial, not necessarily all columns are present. An argument to such a function is written as a list $[T_1, \dots, T_k] \in (2^{\mathcal{S}})^k$, or abbreviated as $\boldsymbol{T}$ for instance. Note that the empty list $[]$ for $k = 0$ is possible. We write $Dom(f)$ for the set of arguments on which f is defined; $f = g$ if f and g have the same domain and agree on that; $f\{\boldsymbol{T} \mapsto U\}$ for the update of f either overwriting a value or extending the domain; and $\{\boldsymbol{T} \mapsto U\}$ for the partial function that contains only this single binding.

Algorithm *MC* takes an HFL1 formula of type τ_k and a list of length k of subsets of the underlying state space $\mathcal{S}$. It returns the semantics of φ applied to these arguments w.r.t. an environment ρ that is given by the global variable *env* which maps each HFL1-variable to a partial function. Note that the semantics of HFL1 is defined using *total* functions though.

Algorithm *MC* simply computes the semantics of a formula recursively according to the definition of HFL1. If the semantics of a fixpoint formula is needed – i.e. the value of the corresponding function on a particular element of its domain – then it performs a fixpoint iteration in the corresponding function space. However, it only computes needed values, hence, localises this fixpoint iteration. It starts with

```
global $env : \mathcal{V} \to \mathcal{D}(\mathcal{S})$

$MC(\varphi, [T_1, ..., T_k]) =$
  case $\varphi$ of $q$                : $L(q)$
            $\neg q$                    : $\mathcal{S} \setminus L(q)$
            $\psi_1 \vee \psi_2$        : $MC(\psi_1, []) \cup MC(\psi_2, [])$
            $\psi_1 \wedge \psi_2$      : $MC(\psi_1, []) \cap MC(\psi_2, [])$
            $\langle a \rangle \psi$    : $\{s \in \mathcal{S} \mid \exists t \in MC(\psi, []) \text{ s.t. } s \xrightarrow{a} t\}$
            $[a]\psi$                   : $\{s \in \mathcal{S} \mid \forall t \in \mathcal{S} : s \xrightarrow{a} t \Rightarrow t \in MC(\psi, [])\}$
            $X$                         : if $env(X)([T_1, ..., T_k]) =$ undef
                                          then let $T :=$ if $fp(X) = \mu$ then $\emptyset$ else $\mathcal{S}$
                                                $env(X) := env(X)\{[T_1, \ldots, T_k] \mapsto T\}$
                                          return $env(X)([T_1, \ldots, T_k])$
            $\neg X$                    : $\mathcal{S} \setminus env(X)([])$
            $\lambda X.\psi$            : $env(X) := \{[] \mapsto T_1\}$
                                          return $MC(\psi, [T_2, ..., T_k])$
            $\psi_1\ \psi_2$            : $MC(\psi_1, [MC(\psi_2, []), T_1, ..., T_k])$
            $\sigma X.\psi$             : if $\sigma = \mu$ then $T := \emptyset$ else $T := \mathcal{S}$
                                          $env(X) := \{[T_1, \ldots, T_k] \mapsto T\}$
                                          repeat
                                            $f := env(X)$
                                            for all $[T'_1, ..., T'_k] \in Dom(env(X))$
                                              $env(X) := env(X)\{[T'_1, ..., T'_k] \mapsto MC(\psi, [T'_1, ..., T'_k])\}$
                                          until $f = env(X)$
                                          return $env(X)([T_1, \ldots, T_k])$
```

Fig. 1. A symbolic model checking algorithm for HFL1

the function that maps the given argument to the initial iteration value – $\emptyset$ or $\mathcal{S}$. If a fixpoint variable is reached during this iteration then the value of the semantics may be required on a different argument. In this case, the algorithm adds the new argument to the domain of this function and includes this in further iterations. This is why a global variable representing the environment is necessary.

We write $MC_\rho(\varphi, [T_1, \ldots, T_k])$ for the result of the call to MC with arguments φ and $[T_1, \ldots, T_k]$ when, at the beginning, the global variable env resembles the environment ρ in the following way: for all $X \in \mathcal{V}$ of type τ_k and all $\boldsymbol{T} \in (2^{\mathcal{S}})^k$ we have

– if X is λ-bound then $env(X) = \{[] \mapsto \rho(X)\}$,
– if X is μ-bound then $\rho(X)(\boldsymbol{T}) \neq \emptyset$ implies $env(X)(\boldsymbol{T}) = \rho(X)(\boldsymbol{T})$,
– if X is ν-bound then $\rho(X)(\boldsymbol{T}) \neq \mathcal{S}$ implies $env(X)(\boldsymbol{T}) = \rho(X)(\boldsymbol{T})$.

Example 3. To illustrate how the algorithm works, consider the formula

$$\varphi_0 \quad := \quad \big(\mu X^{\tau_1}.\lambda Z.Z \vee \bigvee_{a \in \Sigma} X\ [a]Z\big)\ \neg q$$

and the transition system shown on the right side in Fig. 2. Intuitively, φ asserts that there is a sequence of actions s.t. all paths under that sequence lead to

X	$\{3\}$	$\{2,3\}$	$\{1,2,3\}$
0	$\emptyset$		
1	$\{3\}$	$\emptyset$	
2	$\{3\}$	$\{2,3\}$	$\emptyset$
3	$\{2,3\}$	$\{2,3\}$	$\{1,2,3\}$
4	$\{2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
5	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
6	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$

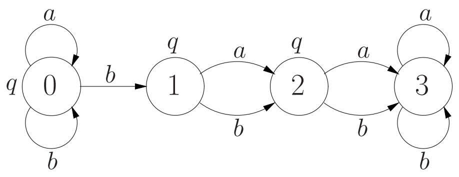


Fig. 2. Algorithm *MC* running on a simple example

a state not satisfying q. States $1, 2, 3$ satisfy this property, state 0 does not. However, the meaning of this formula is irrelevant for the understanding of how it is evaluated by algorithm MC.

The table on the left of Fig. 2 shows the successive calculation of the semantics of the fixpoint formula. Although only two rows need to be stored in each iteration step – the current one and the last one for comparison – we depict all stages in this example for the reader to be be able to follow this step-by-step.

At the beginning, the formula $\neg q$ is evaluated to $\{3\}$. This forms the initial argument in the table. It is to be read as follows: time proceeds line by line from left to right. Each row below the arguments contains a snapshot of the current state at the end of an iteration over the current domain. Note that in general fixpoint approximants cannot easily be read off the table since different columns may be at different stages of approximation. As computation proceeds, arguments are added to the list.

Row 6 then represents a partial function that agrees with the total function that is the semantics of the corresponding fixpoint formula. The return value is the one in the first column – the value of the fixpoint function applied to the original argument.

Theorem 1. *For all transition systems $\mathcal{T}$, all environments ρ and all $\varphi \in$ HFL1 of type* Pr *we have: $MC_\rho(\varphi, []) = [\![\varphi]\!]^{\mathcal{T}}_{\rho}$.*

Proof. By induction on the structure of the formula φ. However, it should be clear that the statement is too weak as an inductive invariant because of subformulas of types other than Pr. Instead, we prove the stronger statement

$$\forall\varphi, \forall\rho, \forall[T_1, \dots, T_k] : MC_\rho(\varphi, [T_1, \dots, T_k]) = [\![\varphi]\!]^{\mathcal{T}}_{\rho}([T_1, \dots, T_k]) \qquad (1)$$

where φ is a (not necessarily closed) formula of type τ_k, ρ is an environment that maps any free variable in φ to a function which in turn is defined on all arguments, and $T_i \subseteq \mathcal{S}$ are subsets of states of the underlying transition system $\mathcal{T}$. We also identify elements of type τ_0, i.e. such subsets, with functions from the singleton domain. Writing $[\![\varphi]\!]^{\mathcal{T}}_{\rho}([])$ rather than $[\![\varphi]\!]^{\mathcal{T}}_{\rho}$ simply spares us some case distinctions in the notations.

The propositional and modal part. Claim (1) is immediately seen to be true for the cases of $\varphi = q$ or $\varphi = \neg q$ for some $q \in \mathcal{P}$. It also follows directly from the hypothesis in the cases $\varphi = \psi_1 \vee \psi_2$, $\varphi = \psi_1 \wedge \psi_2$, $\varphi = \langle a \rangle \psi$ and $\varphi = [a]\psi$. Note that in all these cases, φ must have type τ_0, and so have ψ, ψ_1, ψ_2. Hence, we must have $k = 0$ and the argument list $[T_1, \dots, T_k]$ must in fact be empty.

The statement is also easily seen to be true for the cases of $\varphi = X$ or $\varphi = \neg X$. Note that by assumption, the call of $MC_\rho(X, [T_1, \dots, T_k])$ returns $\rho(X)$. Also remember that negated variables must be λ-bound and therefore of type τ_0.

The functional part. Now consider the case $\varphi = \lambda X.\psi$. Note that φ cannot be of type τ_0, i.e. we have $k \geq 1$. Then $MC_\rho(\varphi, [T_1, \dots, T_k]) = MC_{\rho'}(\psi, [T_2, \dots, T_k])$ with $\rho' := \rho\{X \mapsto T_1\}$. By hypothesis, this is equal to $[\![\psi]\!]^{\mathcal{T}}_{\rho'}([T_2, \dots, T_k])$ which, by β-reduction, is also the same as $[\![\lambda X.\psi]\!]^{\mathcal{T}}_{\rho}([T_1, \dots, T_k])$.

The case of $\varphi = \psi_1\ \psi_2$ is proved analogously. Again, note that here ψ_2 must be of type τ_0.

The only cases posing some difficulties are those of $\varphi = \sigma X.\psi$ for $\sigma \in \{\mu, \nu\}$. Here it is helpful to prove soundness (direction "$\subseteq$" in (1)) and completeness (direction "$\supseteq$") separately. However, the soundness proof for the μ-case is entirely analogous to the completeness proof of the ν-case and vice-versa. Thus, we only present soundness and completeness of the μ-case here.

Soundness of the μ-part. Consider the call $MC_\rho(\mu X.\psi, [T_1, \dots, T_k])$ and the following statement.

$$\forall [T'_1, \dots, T'_k] \in Dom(env(X)) : env(X)([T'_1, \dots, T'_k]) \subseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}([T'_1, \dots, T'_k]) \tag{2}$$

where *env* is assumed to resemble the environment ρ in the way described above. This is in fact an invariant of the `repeat`-loop in Algorithm *MC*. It trivially holds before the loop because $Dom(\rho(X)) = \{[T_1, \dots, T_k]\}$ only, and $env(X)$ maps this tuple to the empty set.

Furthermore, if statement (2) holds at the beginning of one iteration of the `repeat`-loop then it also holds after this iteration. This is simply a consequence of monotonicity, the hypothesis, and the fact that $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}$ is a fixpoint of ψ w.r.t. $\sqsubseteq$: if we have $env(X)([T'_1, \dots, T'_k]) \subseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}([T'_1, \dots, T'_k])$ for all such tuples then, by monotonicity and the definition of the pointwise inclusion ordering, we also have $[\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto env(X)\}} \sqsubseteq [\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto [\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}\}}$. But the latter is equal to $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}$, and the former is, by hypothesis, the content of $env(X)$ on all members of $Dom(env(X))$ at the end of this `repeat`-loop iteration.

This implicitly shows that – on finite transition systems – the loop eventually terminates. Since the domain of $env(X)$ at most grows in each iteration, we have $[T_1, \dots, T_k] \in Dom(env(X))$ at termination point, and the soundness part of (1) immediately follows from the fact that (2) holds at this point.

Completeness of the μ-part. We will prove this part using fixpoint induction. For any set $D \subseteq (2^{\mathcal{S}})^k$ of k-tuples of subsets of the underlying state set $\mathcal{S}$, and two functions $f, g \in \mathcal{D}^k(\mathcal{S})$ we write $f \sqsubseteq_D g$ iff for all $[T'_1, \ldots, T'_k] \in D$: $f([T'_1, \ldots, T'_k]) \subseteq g([T'_1, \ldots, T'_k])$.

Now consider again the call $MC_\rho(\mu X.\psi, [T_1, \ldots, T_k])$. Let $D := Dom(env(X))$ upon termination of the `repeat`-loop. An immediate consequence of the induction hypothesis for ψ is the following:

$$[\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto f\}} \;\sqsubseteq_D\; env(X) \tag{3}$$

for any function f that agrees with $env(X)$ on all arguments in D. This is because the `repeat`-loop is iterated on the whole of D until stability is reached, i.e. until $MC_{\rho\{X \mapsto env(X)\}}(\psi, [T'_1, \ldots, T'_k]) = env(X)([T'_1, \ldots, T'_k])$ holds for all $[T'_1, \ldots, T'_k] \in D$.

We now extend the function $env(X)$ to a function $env^\top(X)$ in the following way.

$$env^\top(X)([T'_1, \ldots, T'_k]) \;:=\; \begin{cases} env(X)([T'_1, \ldots, T'_k]) & \text{, if } [T'_1, \ldots, T'_k] \in D \\ \mathcal{S} & \text{, o.w.} \end{cases}$$

Now note that we have

$$[\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto env^\top(X)\}} \;\sqsubseteq\; env^\top(X)$$

i.e. the function on the right subsumes the one on the left on all arguments from $\mathcal{D}^k(\mathcal{S})$. For arguments in D this is stated in (3) above. For all other arguments this is trivially true by the construction of $env^\top(X)$. But then $env^\top(X)$ is a prefixpoint of ψ and, hence, we have $[\![\mu X.\psi]\!]^{\mathcal{T}}_\rho \sqsubseteq env^\top(X)$. In particular, inclusion holds for all argument tuples in D. Since the domain of $env(X)$ at most grows in each iteration of the `repeat`-loop, we have $[T_1, \ldots, T_k] \in D$ and therefore $[\![\mu X.\psi]\!]^{\mathcal{T}}_\rho([T_1, \ldots, T_k]) \subseteq MC_\rho(\mu X.\psi, [T_1, \ldots, T_k])$ which finishes the proof. □

4 NFA-UNIV as a Model Checking Problem

As stated in the introduction, HFL1 is powerful enough to enable the encoding of various interesting problems as model checking instances. We exemplarily pick the universality problem for non-deterministic finite automata (NFA) to demonstrate how encoding a problem as a model checking instance can lead to an efficient solution.

In the following, an NFA is always of the form $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is the state set, Σ the alphabet, δ the transition relation, q_0 the starting state and F the set of final states. Recall the model in Ex. 3. If the proposition q is interpreted as a flag for being a final state then the whole model can easily be viewed as an NFA. In this context the formula

$$\varphi_0 \;:=\; \big(\mu X^{\tau_1}.\lambda Z.Z \vee \bigvee_{a \in \Sigma} X\ [a]Z\big)\ \neg q$$

translates to "there is a word w, s.t. all states reachable under w are non-final". NFA-UNIV is solved by checking whether or not the starting state satisfies this formula.

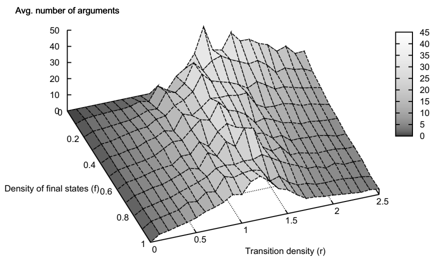


Fig. 3. Number of arguments in function table ($n = 10$)

4.1 Local Fixpoint Computation in Practice

We now give empirical evidence of the benefits of local fixpoint computations and demonstrate that the necessity to compute larger fragments of the complete domain rarely occurs. Algorithm *MC* has been implemented as a prototype in OCaml and run on the Tabakov/Vardi random model for NFAs in order to guarantee a wide spectrum of test cases. Two parameters s and t determine the number of randomly chosen final states and transitions in an NFA for a given number of states n. The ratios $f := \frac{s}{n}$ and $r := \frac{t}{n}$ are called final state density and transition density respectively. For further details see [8]. To perform the universality tests, we fix $n = 10$ and generate 20 random NFAs for each of 250 pairs (r, f) with $0 \leq r \leq 2.5$ and $0 \leq f \leq 1$.

The average number of arguments needed in the fixpoint computation by algorithm *MC* in dependence of (r, f) is depicted in Fig. 3. Note that the number of possible arguments $|2^{\mathcal{S}}|$ is 1024 in this case. Fig. 3 shows that in all cases the algorithm is far away from exhaustive fixpoint calculation on the full argument set $2^{\mathcal{S}}$. Even for the most difficult instances which in our tests are $f = 0.1$ and r between 1.4 and 1.6, the number of needed arguments never gets anywhere near that. The average number of arguments distributed over all 5000 tests is just 13.2 and the highest number of arguments ever measured during the tests is 109.

It is reasonable to assume that the approach of guiding the fixpoint iterations locally through neededness analysis also proves to be successful in other cases (on different formulas) unless the underlying models have been constructed pathologically to enforce an exponential behaviour.

4.2 Optimising Algorithm *MC* w.r.t. a Fixed Formula

There are still several standard performance enhancements available, e.g. acceleration of the fixpoint computation by exploiting monotonicity, in order to optimise this algorithm.

However, we need to observe that algorithm *MC* will be used on fixed formulas in most cases. In many verification tasks the property to be checked is fixed while the models change. This holds especially for non-regular properties since non-regularity often eliminates dependence on model sizes, etc. It is therefore much more beneficial to regard *MC* as a *template* for specialised cases rather than a general algorithm for all kinds of verification purposes. Model checking a fixed formula bears a higher potential for algorithm optimisations which possibly cannot be achieved for varying formulas.

Consider the algorithm's behaviour on the formula of Ex. 3 as depicted in the table there. If we follow the succession of the fixpoint iteration closely, a simple pattern can be observed: the iterated function $\lambda Y.Y \vee \bigvee_{a\in\Sigma} X\ [a]Y$ takes an argument (initially the set $[\![\neg q]\!]^{\mathcal{A}}$) and returns its union with the set of its recursive $[a]$-predecessors for all $a \in \Sigma$. But this set is exactly the union of the elements of $Dom(X)$, each of them the result of a single $[a]Y$ computation step. So the return value does not provide any additional information if the set of needed arguments is known. Furthermore, since only a union operation is performed, it suffices to keep track of $\subseteq$-maximal sets of arguments. This insight immediately leads to an optimisation by discarding all redundant information. It is obviously not necessary to protocol all these values in the fixpoint iterations – when in the end all we want to know is whether or not the initial automaton state is included in the union over all arguments. It suffices to iterate this schema until no more arguments enter the table, and then to form their unions. This, however, means that, by monotonicity of the $[a]$-operators, one can always discard the larger of two arguments that are comparable w.r.t. $\subseteq$ which leads to the idea of storing $Dom(X)$ as an *antichain.*

An antichain over an NFA $\mathcal{A}$ is a set $\mathcal{C}$ of pairwise incomparable (w.r.t. set inclusion) sets of states of $\mathcal{A}$. These antichains form a complete lattice when equipped with the following order.

$$\mathcal{C} \sqsubseteq \mathcal{C}' \quad \text{iff} \quad \forall C \in \mathcal{C}\ \exists C' \in \mathcal{C}' \text{ s.t. } C \subseteq C'$$

This naturally induces a notion of supremum $\mathcal{C} \sqcup \mathcal{C}'$ as the smallest antichain (w.r.t. $\sqsubseteq$) which contains both $\mathcal{C}$ and $\mathcal{C}'$.

The basic principle of the optimization is to populate an antichain with sets of states which uphold the possibility of generating a word that is not included in the language of the automaton. This can be achieved by loosely speaking applying the modal $[a]$-operator (for all $a \in \Sigma$) to its elements and minimizing the resulting set to an antichain. More formally, define the following monotone operation on antichains:

$$CPre(\mathcal{C}) \;:=\; \lceil\{S \subseteq Q \mid \exists T \in \mathcal{C}\ \exists a \in \Sigma \text{ s.t. } S = [\![[a]X]\!]^{\mathcal{A}}_{\{X\mapsto T\}}\}\rceil$$

where the $\lceil\cdot\rceil$ operator discards all sets which are subsumed by another set in this set of sets – i.e. it makes an antichain of the expression on the right-hand side.

Henzinger et al. show how to characterise NFA-UNIV using least fixpoints in antichain lattices.

Proposition 1. [10, Thm. 2] *Let $\mathcal{A}$ be an NFA over the alphabet Σ with state set Q, initial state q_0 and final states F. Then $L(\mathcal{A}) \neq \Sigma^*$ iff $\{\{q_0\}\} \sqsubseteq \bigsqcap\{\mathcal{C} \mid CPre(\mathcal{C}) \sqcup \{Q \setminus F\} \sqsubseteq \mathcal{C}\}$.*

Of course, the least fixpoint can be computed by a straight-forward fixpoint iteration: Define $\mathcal{C}^0 := \{\emptyset\}$ and $\mathcal{C}^i := \mathit{CPre}(\mathcal{C}^{i-1}) \sqcup \{Q \setminus F\}$. The following table compares in parallel two runs of MC and the antichain method on Ex. 3:

X	$\{3\}$	$\{2,3\}$	$\{1,2,3\}$
0	$\emptyset$		
1	$\{3\}$	$\emptyset$	
2	$\{3\}$	$\{2,3\}$	$\emptyset$
3	$\{2,3\}$	$\{2,3\}$	$\{1,2,3\}$
4	$\{2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
5	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
6	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$

$$\begin{aligned}
\mathcal{C}^0 &:= \{\emptyset\} \\
\mathcal{C}^1 &:= \mathit{CPre}(\mathcal{C}^0) \sqcup \{Q \setminus F\} = \{\{3\}\} \\
\mathcal{C}^2 &:= \mathit{CPre}(\mathcal{C}^1) \sqcup \{Q \setminus F\} = \{\{2,3\}\} \\
\mathcal{C}^3 &:= \mathit{CPre}(\mathcal{C}^2) \sqcup \{Q \setminus F\} = \{\{1,2,3\}\} \\
\mathcal{C}^4 &:= \mathit{CPre}(\mathcal{C}^3) \sqcup \{Q \setminus F\} = \{\{1,2,3\}\}
\end{aligned}$$

The cost reduction of the antichain method is established by the fact that it simply computes $\lceil \mathit{Dom}(X) \rceil$, i.e. the antichain of the currently present arguments. One can show that $\lceil \mathit{Dom}(X^i) \rceil = \mathcal{C}^{i+1}$, where $\mathit{Dom}(X^i)$ is the currently needed domain of the ith fixpoint approximation w.r.t. a given argument and a partial evaluation according to MC.

It turns out that the result of this optimisation is exactly the method devised by Henzinger et al. in [10]. Their tool shows a very good performance on the universality test for NFAs and does apparently outperform the classical powerset construction by several orders of magnitude.

5 Conclusion and Outlook

We have presented a model checking algorithm for HFL1. This extends the scope of properties which can automatically be checked on finite state systems way beyond that of regular ones whilst keeping the complexity at most singly exponential in the size of the system. In order to avoid exponential best-case behaviour we suggest to localise computations of fixpoints in function spaces.

This algorithm fully supports symbolic model checking using a BDD library. A prototypical implementation has been created which shows the gain of local fixpoint computations in this setting. This approach is most successful on instances with fixed formulas. This allows to optimise algorithm MC further w.r.t. that particular formula as seen with the NFA-UNIV example where a rigorous optimisation of algorithm MC yields the competitive antichain algorithm of Henzinger et al.

5.1 Other Hard Decision Problems as Model Checking Instances

By not just restricting the term "model checking" to a method used in automatic program verification but understanding it as a general logic problem we can obtain BDD-based algorithms for various other problems as well. Note that NFA-UNIV is PSPACE-complete, and it is therefore reasonable to try to encode the standard PSPACE-complete problem QBF as an HFL1 model checking problem.

It is well-known that every quantified Boolean formula can be put into prenex CNF normal form $Q_1 x_1 . \dots Q_n x_n . \bigwedge_i \bigvee_{j_i} l_{i,j_i}$ with the $Q_k \in \{\exists, \forall\}$, and the l_{i,j_i}

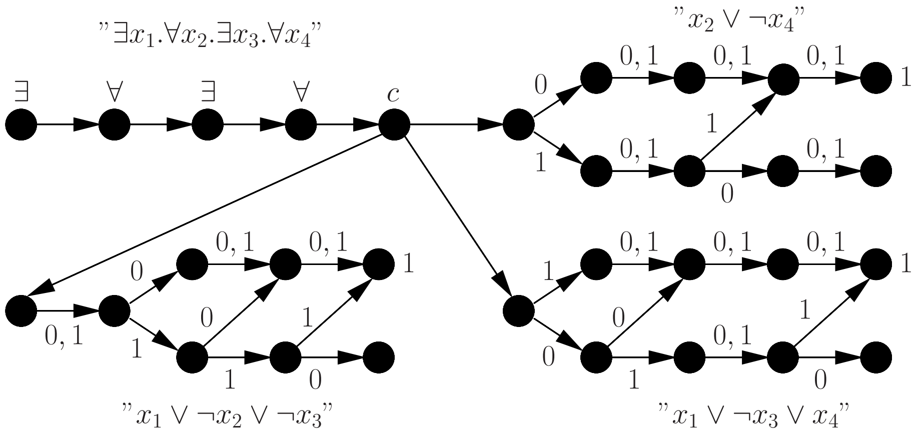


Fig. 4. A transition system representation of a QBF formula

literals over the variables $x_1, \ldots, x_n$. The problem QBF is to decide whether or not such a formula evaluates to 1 under the usual interpretation of the Boolean operators and the quantifiers over the domain $\{0, 1\}$.

With each QBF formula Φ we associate a loop-free transition system $\mathcal{T}_\Phi$ which is exemplarily shown in Fig. 4 for $\Phi = \exists x_1.\forall x_2.\exists x_3.\forall x_4.(x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$. It uses atomic propositions $\exists$, $\forall$ to mark the type of quantification over a variable, c to indicate the branching into the different clauses, and 1 to mark the value of a clause under an assignment valuation given by a path through each clause's component. Its actions are 0 and 1 for representing variable values, and an anonymous one for branching into different clauses and for separating the quantifiers in the prefix.

Evaluation to 1 of Φ can now be expressed in HFL1 as follows.

$$\begin{aligned}\varphi_{QBF} \;:=\; \Big(\mu X^{\tau_1}.\lambda Z.\big(c \to [-]Z\big) \wedge \big(\exists \;\to\; \langle -\rangle(X\ \langle 0\rangle Z) \vee \langle -\rangle(X\ \langle 1\rangle Z)\big) \wedge \\ \big(\forall \;\to\; \langle -\rangle(X\ \langle 0\rangle Z) \wedge \langle -\rangle(X\ \langle 1\rangle Z)\big)\Big)\ 1\end{aligned}$$

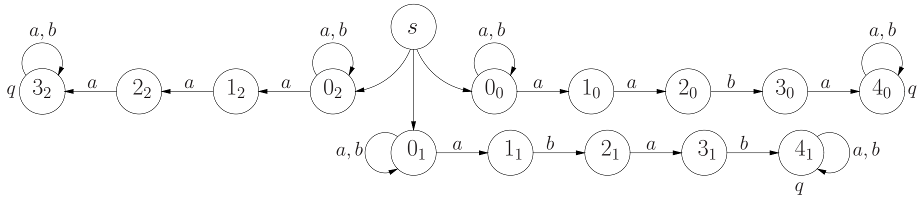
Again, φ_{QBF} does not depend on the underlying QBF formula Φ. It is therefore possible to obtain a (BDD-based – if desired) QBF solver by analysing the behaviour of algorithm *MC* on φ_{QBF} and specialised transition systems $\mathcal{T}_\Phi$. For example, it is not hard to see that the fixpoint iteration always terminates after a number of steps given by the length of the quantifier prefix. It can therefore be made explicit through a `for`-loop. Furthermore, antichains can also be used to replace the arguments of the function table. Preliminary results show that this is far from yielding a competitive QBF solver. However, it may be interesting to investigate combinations of this bottom-up approach with existing solvers that mostly work top-down.

Algorithms for other problems can also be obtained by instantiating the template *MC* with a particular formula: satisfiability of modal logic where it remains to compare the result to the BDD-based algorithm by Pan et al. [7]; various problems from automata-theory like emptiness of alternating finite automata; etc. Due to space restrictions we will elaborate on details of those elsewhere.

5.2 Encoding Optimisation Problems

Some optimisation problems that require more than a yes/no answer can also be dealt with using an extension of algorithm *MC* that keeps track of parts of the solution to be computed. We sketch a new algorithm for the Shortest Common Supersequence problem (SCS): given a set $\{w_1, \dots, w_n\}$ of finite words of some alphabet Σ, find a shortest $v \in \Sigma^*$ that contains all w_i as subwords. The algorithm is obtained from the template *MC* using an antichain optimisation as in the case of NFA-UNIV.

The first step consists of building a transition system $\mathcal{T}$, here depicted for the words $\{aaba, abab, aaa\}$.



Next, consider the HFL1 formula $\varphi_{SCS} := \big(\mu X^{\tau_1}.\lambda Z.[-]Z \vee \bigvee_{a\in\Sigma} X\ \langle a\rangle Z\big)\ q$. Each state in $\mathcal{T}$ satisfies φ_{SCS} which only reflects the fact that for every finite set of words there is a word containing all of them. However, suppose the arguments in the table for the fixpoint iteration in this formula are annotated in the following way: the initial argument receives the annotation ϵ, and if an argument Z with annotation w causes another argument to be created in the table through the recursive call of $X\ \langle a\rangle Z$ then the new argument receives the annotation aw.

Now note the apparent similarity of this formula with the one from Ex. 3 expressing NFA-UNIV. In both cases the subformulas $X\ \psi(Z)$ only occur under a disjunction. Hence, the argument row of the function table can again be optimised into an antichain, and the evaluation of the formula can be regarded as a fixpoint iteration in an antichain lattice. It terminates when the topmost state of $\mathcal{T}$ occurs in an element of the current antichain, and that element's annotation is the solution to the SCS problem.

The computation of the solution *aaabab* using annotated antichains is found as follows. Let $I := \{4_0, 4_1, 3_2\}$. For a set S we write S_I^w to abbreviate $(S \cup I)^w$ where the superscript simply denotes the word annotation of this set.

$$
\begin{aligned}
\mathcal{C}_0 &:= \{I^\epsilon\} \\
\mathcal{C}_1 &:= \{\{2_2, 3_0\}_I^a, \{3_1\}_I^b\} \\
\mathcal{C}_2 &:= \{\{2_2, 2_1, 3_0\}_I^{ab}, \{2_2, 1_2, 3_0\}_I^{aa}, \{3_1, 2_0\}_I^{ba}\} \\
\mathcal{C}_3 &:= \{\{2_2, 2_1, 3_0, 1_0\}_I^{aba}, \{2_2, 1_2, 0_2, 3_0\}_I^{aaa}, \{3_1, 1_1, 2_0\}_I^{bab}\} \\
\mathcal{C}_4 &:= \{\{2_2, 2_1, 0_1, 3_0, 1_0\}_I^{abab}, \{2_2, 1_2, 0_2, 3_0\}_I^{aaaa}, \{2_2, 1_2, 3_0, 0_0\}_I^{aaba}, \\
&\qquad \{0_2, 3_1, 2_0\}_I^{baaa}, \{3_1, 1_1, 2_0\}_I^{baba}\}
\end{aligned}
$$

$$\begin{aligned}\mathcal{C}_5 &:= \{\{\ldots\}_I^{ababa}, \{\ldots\}_I^{abaaa}, \{\ldots\}_I^{aaaaa}, \{2_2, 1_2, 0_1, 3_0, 0_0\}_I^{aabab},\\ &\quad \{\ldots\}_I^{baaba}, \{\ldots\}_I^{baaaa}, \{\ldots\}_I^{babab}\}\\ \mathcal{C}_6 &:= \{\ldots, \{2_2, 1_2, 0_2, 0_0, 0_1\}_I^{aaabab}, \ldots\}\end{aligned}$$

Finally, since a set containing $\{0_0, 0_1, 0_2\}$ has been found, s is included in the next iteration, and the solution is the annotation of this witnessing set.

We will compare this new algorithm for SCS to existing ones and investigate its use in bio-informatics for example elsewhere.

References

1. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004)
2. Axelsson, R., Lange, M., Somla, R.: The complexity of model checking higher-order fixpoint logic. In: Logical Methods in Computer Science (accepted for publication, 2007)
3. Emerson, E.A.: Uniform inevitability is tree automaton ineffable. Information Processing Letters 24(2), 77–79 (1987)
4. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proc. 32nd Symp. on Foundations of Computer Science, San Juan, Puerto Rico, pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)
5. Jørgensen, N.: Finding fixpoints in finite function spaces using neededness analysis and chaotic iteration. In: LeCharlier, B. (ed.) SAS 1994. LNCS, vol. 864, pp. 329–345. Springer, Heidelberg (1994)
6. Müller-Olm, M.: A modal fixpoint logic with chop. In: Meinel, C., Tison, S. (eds.) STACS 99. LNCS, vol. 1563, pp. 510–520. Springer, Heidelberg (1999)
7. Pan, G., Sattler, U., Vardi, M.Y.: BDD-based decision procedures for the modal logic K. Journal of Applied Non-Classical Logics 16(1-2), 169–208 (2006)
8. Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 396–411. Springer, Heidelberg (2005)
9. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 512–528. Springer, Heidelberg (2004)
10. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: A new algorithm for checking universality of finite automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 17–30. Springer, Heidelberg (2006)