An Incremental Technique for Automata-Based Decision Procedures

Gulay Unel and David Toman

D.R.Cheriton School of Computer Science
University of Waterloo
{gunel,david}@cs.uwaterloo.ca

Abstract. Automata-based decision procedures commonly achieve optimal complexity bounds. However, in practice, they are often outperformed by sub-optimal (but more local-search based) techniques, such as tableaux, on many practical reasoning problems. This discrepancy is often the result of automata-style techniques global approach to the problem and the consequent need for constructing an extremely large automaton. This is in particular the case when reasoning in theories consisting of large number of relatively simple formulas, such as descriptions of database schemes, is required. In this paper, we propose techniques that allow us to approach a μ-calculus satisfiability problem in an incremental fashion and without the need for re-computation. In addition, we also propose heuristics that guide the problem partitioning in a way that is likely to reduce the size of the problems that need to be solved.

1 Introduction

Propositional μ-calculus, thanks to its high expressive power, is often considered one of the *lingua franca* logical formalism among logics with EXPTIME decision procedures. Indeed, many other modal, dynamic, temporal, and description logics have been shown to be relatively easily encodable in μ-calculus [8,16,24].

The key technique to showing decidability and complexity bounds for μ-calculus is based on capturing *the language of models* of a given formula using an automaton constructed from the formula—usually an *alternating parity automaton*—that accepts infinite tree models of the formula [25,26,27]. Hence, testing for satisfiability reduces to testing for non-emptiness of an appropriate automaton.

In practice, however, automata-based decision procedures do not enjoy the success predicted by the accompanying theory. Indeed, in many cases, theoretically sub-optimal approaches, such as the use of tableaux equipped with appropriate *blocking conditions* that prevent infinite expansions, are more successful [1,12]. This rather surprising observation can be traced to severe difficulties in implementing automata-based decision procedures, in particular when inherently infinite models are considered. For example, the emptiness test for alternating parity automaton, in particular when based on Safra's determinization approach [22,23], is rather difficult to implement. This issue, for μ-calculus

F. Pfenning (Ed.): CADE 2007, LNAI 4603, pp. 100–115, 2007.

formulas, was addressed by using simpler *Safraless* decision procedures based on transforming an alternating parity automaton to a non-deterministic Büchi automaton while preserving emptiness [19].

However, even this improvement does not yield a practical reasoning procedure. The difficulties *inherent in the automata-based approaches* are especially apparent when determining *logical consequences* of moderately large *theories* of the form $\{\varphi_1, \ldots \varphi_n\} \models \varphi$, are considered. Commonly, more local search techniques applied to this problem try to discover an inconsistency in the set $\{\varphi_1, \ldots \varphi_n, \neg\varphi\}$, which in practice rarely involves all the formulas φ_i in the input. Hence, the inconsistency can often be detected much more efficiently than using the automata-theoretic method which is constructing the automaton for the formula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \wedge \neg\varphi$ and then checking for its emptiness. This problem manifests itself in many important settings, in which theories that describe system behavior use a large number of relatively simple constraints, such as database schemes or UML diagrams specified using, e.g., an appropriate description logic [2,7].

In this paper, we explore techniques that attempt to remedy the above difficulties by proposing an incremental and interleaved approach to constructing the automaton corresponding to the logical implication problem while simultaneously testing for satisfiability of the so far constructed fragments. The main contributions of this paper are as follows:

- we show how the decision problem can be split into a sequence of simpler problems,
- we show that in this incremental process, the larger problems can be constructed from the simpler ones, hence avoiding unnecessary recomputation, and
- we show how top-down query evaluation techniques enhanced with memoing can be used to drive the incremental computation.

The rest of the paper is organized as follows: Section 2 provides the necessary definitions and background, Section 3 introduces the incremental approach and outlines the main results, Section 4 discusses heuristics and optimizations of the proposed algorithm, and Section 6 concludes outlining directions of further research.

2 Preliminaries

In this section, we provide definitions needed for the technical development in the rest of the paper.

2.1 μ-Calculus

The propositional μ-calculus is a propositional modal logic augmented with least and greatest fixpoint operators [16]. The syntax of μ-calculus [4] is given below:

Definition 1. *Let* Var *be an (infinite) set of variable names, typically named* $X, Y, Z, \ldots$*;* Prop *a set of atomic propositions, typically named* $P, Q, \ldots$*; and* $\mathcal{L}$ *a set of labels,* $a, b, \ldots$*. The set of* L_μ *formulas (with respect to* Var*,* Prop*,* $\mathcal{L}$*) is defined as follows:*

- $P \in$ Prop *and* $Z \in$ Var *are formulas.*
- *If* ϕ_1 *and* ϕ_2 *are formulas, so is* $\phi_1 \wedge \phi_2$*.*
- *If* ϕ *is a formula, so are* $[a]\phi$*,* $\neg\phi$*, and* $\nu Z.\phi$ *provided that every free occurrence of* Z *in* ϕ *occurs positively.*

In the rest of the paper we use derived operators, e.g., $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\langle a\rangle\phi$ for $\neg[a]\neg\phi$, $\mu Z.\phi(Z)$ for $\neg\nu Z.\neg\phi(\neg Z)$, $[K]\phi$ for $\bigwedge_{a\in K}[a]\phi$, $[-]\phi$ for $[\mathcal{L}]\phi$, etc.

Formulas of L_μ are interpreted with respect to labeled transition systems over Prop in which nodes are labeled by propositional assignments and edges by elements of $\mathcal{L}$; for full definition see [4].

2.2 Alternating Automata

Satisfiable L_μ formulas enjoy the *tree model* property. This property provides a link to automata theory: satisfiability of a L_μ formula is equivalent to checking whether a corresponding tree automaton that accepts tree models of the formula is non-empty.

Definition 2. *Given a set* D *of directions, a* D*-tree is a set* $T \subseteq D^*$ *such that if* $x \cdot c \in T$ *(an extension of* x *with* c*), where* $x \in D^*$ *and* $c \in D$*, then also* $x \in T$*. If* $T = D^*$*, we say that* T *is a full* D*-tree. The empty word* ϵ *is the root of* T *and the elements of* T *are called nodes. A path* π *of a tree* T *is a set* $\pi \subseteq T$ *such that* $\epsilon \in \pi$ *and for every* $x \in \pi$ *either* x *is a leaf or there exists a unique* c *such that* $x \cdot c \in \pi$*. Given an alphabet* Σ*, a* Σ*-labeled* D*-tree is a pair* $\langle T, \tau\rangle$ *where* T *is a tree and* $\tau : T \to \Sigma$ *maps each node of* T *to a letter in* Σ*.*

For a set X*,* $\mathcal{B}^+(X)$ *is the set of positive Boolean formulas over* X*; for a set* $Y \subseteq X$ *and a formula* $\phi \in \mathcal{B}^+(X)$*, we say that* Y *satisfies* ϕ *iff assigning* **true** *to elements in* Y *and assigning* **false** *to elements in* $X \setminus Y$ *makes* ϕ *true. An alternating tree automaton is* $A = \langle \Sigma, D, Q, q_i, \delta, \alpha\rangle$*, where* Σ *is the input alphabet,* D *is a set of directions,* Q *is a finite set of states,* $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$ *is a transition function,* $q_i \in Q$ *is an initial state, and* α *specifies the acceptance condition.*

An alternating automaton A *runs on* Σ*-labeled full* D*-trees. A run of* A *over a* Σ*-labeled* D*-tree* $\langle T, \tau\rangle$ *is a* $(T \times Q)$*-labeled tree* $\langle T_r, r\rangle$ *such that:*

1. $\epsilon \in T_r$ *and* $r(\epsilon) = \langle \epsilon, q_i\rangle$*.*
2. *For every* $y \in T_r$ *such that* $r(y) = \langle x, q\rangle$ *there is a set*

$$\{(c_0, q_0), (c_1, q_1), \ldots (c_{n-1}, q_{n-1})\} \subseteq D \times Q$$

that satisfies $\delta(q, \tau(x))$*, and for all* $0 \leq j < n$*,* $y \cdot j \in T_r$*,* $r(y \cdot j) = \langle x \cdot c_j, q_j\rangle$*.*

A run $\langle T_r, r\rangle$ is accepting if all its infinite paths satisfy an acceptance condition. The set of states on a path $\pi \subseteq T_r$ that appear infinitely often is denoted with $\mathsf{inf}(\pi)$ where $\mathsf{inf}(\pi) \subseteq Q$ and $q \in \mathsf{inf}(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. The types of acceptance conditions are defined as follows:

- *A path π satisfies Büchi acceptance condition $\alpha \subseteq Q$ if $\mathsf{inf}(\pi) \cap \alpha \neq \emptyset$.*
- *A path π satisfies co-Büchi acceptance condition $\alpha \subseteq Q$ if $\mathsf{inf}(\pi) \cap \alpha = \emptyset$.*
- *A path π satisfies parity acceptance condition $\alpha = \{F_1, F_2, \ldots, F_h\}$ with $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_h = Q$ if the minimal index i for which $\mathsf{inf}(\pi) \cap F_i \neq \emptyset$ is even. The number h of sets in α is called the index of the automaton.*

An automaton accepts a tree if there exists a run that accepts it. The set of all Σ-trees that are accepted by A is denoted by $\mathcal{L}(A)$. An alternating automaton is:

- *nondeterministic: if the formulas (c_1, q_1) and (c_2, q_2) appear in δ and are conjunctively related, then $c_1 \neq c_2$,*
- *universal: if all the formulas that appear in δ are conjunctions of atoms in $D \times Q$,*
- *deterministic: if it satisfies the conditions for being nondeterministic and universal at the same time.*

The connection between L_μ formulas and alternating automata is captured by the following theorem [9,13,27].

Theorem 1. *Let $\varphi \in L_\mu$. Then there is an alternating parity tree automaton A_φ that can be constructed effectively from φ, such that the language of trees accepted by A_φ is the set of tree models of φ.*

Hence, it remains to solve the emptiness problem for alternating automata to decide the satisfiability of μ-calculus formulas. Logical implication problems can be solved by using the associated satisfiability problems (possibly with the help of the greatest fixpoint operator when *global axioms* are needed).

2.3 From APT to NBT Via UCT

The standard approach for checking the emptiness of an alternating parity tree automaton (APT) involves Safra's construction [22] which is complicated and not very suitable for efficient implementation. An alternative approach to this problem has been proposed by Vardi and Kupferman [19] and involves the following steps:

1. Translate the APT A representing a μ-calculus formula φ to a Universal Co-Büchi Tree Automaton (UCT) A',
2. Translate the UCT A' to a Non-deterministic Büchi Tree Automaton (NBT) A'', and
3. Check for emptiness of A''.

The above transformations only preserve emptiness for the automata, not the actual languages of trees accepted. This is, however, sufficient for deciding satisfiability. We modify this procedure to operate in an incremental fashion when the original alternating automaton represents a conjunction of L_μ formulas. First, we outline the two main steps in the original construction [19]:

From APT to UCT. Consider an APT $A = \langle \Sigma, D, Q, q_i, \delta, \alpha \rangle$, where $\delta : Q \times \Sigma \to \mathcal{B}^+(D \times Q)$. A *restriction* of δ is a partial function $\eta : Q \to 2^{D \times Q}$. A restriction η is relevant to $\sigma \in \Sigma$ if for all $q \in Q$ for which $\delta(q, \sigma)$ is satisfiable, the set $\eta(q)$ satisfies $\delta(q, \sigma)$. Let R be the set of restrictions of δ.

For $A = \langle \Sigma, D, Q, q_i, \delta, \alpha \rangle$ with $\alpha = \{F_1, F_2, \ldots, F_{2h}\}$, and $F_0 = \emptyset$, the UCT is defined as $A' = \langle \Sigma', D, Q \times \{0, \ldots, h-1\}, \langle q_i, 0 \rangle, \delta', \alpha' \rangle$ where:

- $\Sigma' \subseteq \Sigma \times R$ such that η is relevant to σ for all $\langle \sigma, \eta \rangle \in \Sigma'$.
- For every $q \in Q$, $\sigma \in \Sigma$, and $\eta \in R$:
  - $\delta'(\langle q, 0 \rangle, \langle \sigma, \eta \rangle) = \bigwedge_{0 \leq i < h} \bigwedge_{(c,s) \in (\eta(q) \setminus (D \times F_{2i}))} (c, \langle s, i \rangle)$.
  - For every $1 \leq i < h$, $\delta'(\langle q, i \rangle, \langle \sigma, \eta \rangle) = \bigwedge_{(c,s) \in (\eta(q) \setminus (D \times F_{2i}))} (c, \langle s, i \rangle)$.
- $\alpha' = \bigcup_{0 \leq i < h} (F_{2i+1} \times \{i\})$

Intuitively, the nondeterminism in A is removed in A' since Σ' contains all the pairs $\langle \sigma, \eta \rangle$ for which η is relevant to σ (η chooses from all the possible sets of atoms that satisfy δ). The automaton A' consists of h copies of A such that the ith copy checks if a path in a run of A' visits F_{2i} only finitely often then it also visits F_{2i+1} only finitely often by making sure that the run stays in the ith copy unless it has to move to a state from F_{2i}.

From UCT to NBT. Let $A' = \langle \Sigma', D', Q', q'_i, \delta', \alpha' \rangle$, and let $k = (2n!)n^{2n}3^n (n+1)/n!$. Let $\mathcal{R}$ be the set of functions $f : Q' \to \{0, \ldots k\}$ in which $f(q)$ is even for all $q \in \alpha'$. For $g \in \mathcal{R}$, let $odd(g) = \{q : g(q) \text{ is odd}\}$. The definition of $A'' = \langle \Sigma', D', Q'', q''_i, \delta'', \alpha'' \rangle$ is given as follows:

- $Q'' = 2^{Q'} \times 2^{Q'} \times \mathcal{R}$
- $q''_i = \langle \{q'_i\}, \emptyset, g_0 \rangle$, where g_0 maps all states to k.
- For $q \in Q'$, $\sigma \in \Sigma'$, and $c \in D'$, let $\gamma'(q, \sigma, c) = \delta'(q, \sigma) \cap (\{c\} \times Q)$. For two functions g and g' in $\mathcal{R}$, a letter σ, and direction $c \in D'$, we say that g' *covers* $\langle g, \sigma, c \rangle$ if for all q and q' in Q', if $q' \in \gamma'(q, \sigma, c)$. then $g'(q') \leq g(q)$. Then for all $\langle S, O, g \rangle \in Q''$ and $\sigma \in \Sigma'$, δ'' is defined as follows:
  - If $O \neq \emptyset$ then $\delta''(\langle S, O, g \rangle, \sigma)$
$$= \bigwedge_{c \in D} \bigvee_{g_c \ covers \ \langle g, \sigma, c \rangle} \langle \gamma'(S, \sigma, c), \gamma'(O, \sigma, c) \setminus odd(g_c), g_c \rangle$$
  - If $O = \emptyset$ then $\delta''(\langle S, O, g \rangle, \sigma)$
$$= \bigwedge_{c \in D} \bigvee_{g_c \ covers \ \langle g, \sigma, c \rangle} \langle \gamma'(S, \sigma, c), \gamma'(S, \sigma, c) \setminus odd(g_c), g_c \rangle$$
- $\alpha'' = 2^{Q'} \times \{\emptyset\} \times \mathcal{R}$.

Intuitively, the automaton A'' is the result of a subset construction applied to A' such that for a run of A' that satisfies a particular co-Büchi condition it guesses the possible runs that satisfy its dual Büchi condition. The emptiness problem for NBT is much simpler than the emptiness problem for APT which is shown to be solved symbolically in quadratic time [17].

3 Incremental Approach to Satisfiability of Conjunctions

In this section, we provide the main contribution of this paper: a decomposition technique for the APT to NBT construction based on conjunctive formulas and, in turn, an incremental algorithm for checking the emptiness of an APT A for a formula φ. We also outline a top-down approach for checking the emptiness of the associated NBT.

Assume that we have a conjunctive formula $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ and φ is represented by an APT $A = \langle \Sigma, D, Q, q_i, \delta, \alpha \rangle$. We decompose the APT to NBT translation in such a way that we do not need to construct the complete automaton for φ and we can stop checking for emptiness if $\varphi' = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$ for $k \leq n$ is unsatisfiable. Otherwise we are able to reuse the facts we computed for φ' in the emptiness check of φ.

The incremental technique first constructs an automaton A_1 for φ_1 and checks for its emptiness, if A_1 is empty then the procedure stops. Otherwise it continues with automata for formulas $\varphi_1 \wedge \varphi_2$, ..., $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ applying the same technique and reusing the automaton computed in step i for computing the automaton in step $i + 1$ as it is shown in Figure 1.

3.1 Decomposition of the APT to NBT Translation

In this section, we describe the proposed decomposition technique for a conjunction of formulas of the form $\varphi = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$. We know that there is an APT $A = \langle \Sigma, D, Q, q_i, \delta, \alpha \rangle$ that accepts tree models of φ. To define this automaton, we need the following auxiliary definition:

Definition 3. *The closure of a formula ϕ, $\mathsf{cl}(\phi)$ is the smallest set of formulas that satisfies the following:*

- $\phi \in \mathsf{cl}(\phi)$.
- *If* $\phi_1 \wedge \phi_2 \in \mathsf{cl}(\phi)$, *then* $\phi_1 \in \mathsf{cl}(\phi)$ *and* $\phi_2 \in \mathsf{cl}(\phi)$.
- *If* $[a]\psi$ *or* $\neg\psi \in \mathsf{cl}(\phi)$ *then* $\psi \in \mathsf{cl}(\phi)$.
- *If* $\nu Z.\psi \in \mathsf{cl}(\phi)$, *then* $\psi(\nu Z.\psi) \in \mathsf{cl}(\phi)$ *and* $\psi \in \mathsf{cl}(\phi)$.

Now we define an alternating automaton $A_k = \langle \Sigma_k, D, Q_k, q_{i_k}, \delta_k, \alpha_k \rangle$ for a subformula $\varphi' = \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$ of φ as follows:

- $\Sigma_k = 2^{AP_k}$ where AP_k is the set of atomic propositions in φ',
- $q_{i_k} = \varphi'$,
- $Q_k = \mathsf{cl}(\varphi')$,
- for all $\sigma \in \Sigma_k$, $\delta(q, \sigma) \in \delta_k$ iff $q \in Q_k$, and
- $\alpha_k = \{F_1 \cap Q_k, F_2 \cap Q_k, \ldots, F_{2h} \cap Q_k\}$.

Emptiness of A_k implies emptiness of A and, in turn, the unsatisfiability of the original formula φ, as A_k represents a subformula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$ of φ. Hence, we can stop checking for the emptiness of the automaton A early: whenever we reach an automaton A_k that is empty. Otherwise we use the following theorem to *extend* A_k to A_{k+1} without the need to recompute all the transitions from scratch:

Theorem 2. *Let* $A'_k = \langle \Sigma'_k, D, Q_k \times \{0, \ldots h-1\}, \langle q_{i_k}, 0\rangle, \delta'_k, \alpha'_k\rangle$ *be the UCT translation of* A_k*, and* $A' = \langle \Sigma', D, Q \times \{0, \ldots h-1\}, \langle q_i, 0\rangle, \delta', \alpha'\rangle$ *be the UCT translation of* A*.*

Then for every $q_k \in Q_k$*,* $\sigma_k \in \Sigma_k$*,* $\eta_k \in R_k$*,* $\eta_l \in R_l$*:*
$\delta'(\langle q_k, i\rangle, \langle \sigma_k, \eta_k \cup \eta_l\rangle) = \delta'_k(\langle q_k, i\rangle, \langle \sigma_k, \eta_k\rangle)$ *for all* $0 \leq i < h$ *where* R_k *is the set of restrictions* $\eta_k : Q_k \to 2^{D \times Q_k}$ *such that for all* $\langle \sigma_k, \eta_k\rangle \in \Sigma'_k$*,* η_k *is relevant to* σ_k*, and* R_l *is the set of restrictions* $\eta_l : Q \setminus Q_k \to 2^{D \times Q}$*.*

Proof. For every $q_k \in Q_k$, $\sigma_k \in \Sigma_k$, $\eta_k \in R_k$, $\eta_l \in R_l$:

- $\begin{aligned}\delta'(\langle q_k, 0\rangle, \langle \sigma_k, \eta_k \cup \eta_l\rangle) &= \textstyle\bigwedge_{0 \leq i < h} \bigwedge_{(c,s) \in (\eta_k(q_k) \cup \eta_l(q_k)) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \\ &= \textstyle\bigwedge_{0 \leq i < h} \bigwedge_{(c,s) \in (\eta_k(q_k) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \wedge \\ &\qquad \textstyle\bigwedge_{0 \leq i < h} \bigwedge_{(c,s) \in (\eta_l(q_k) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \\ &= \textstyle\bigwedge_{0 \leq i < h} \bigwedge_{(c,s) \in (\eta_k(q_k) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \\ &= \textstyle\bigwedge_{0 \leq i < h} \bigwedge_{(c,s) \in (\eta_k(q_k) \setminus (D \times (F_{2i} \cap Q_k)))} (c, \langle s, i\rangle) \\ &= \delta'_k(\langle q_k, 0\rangle, \langle \sigma_k, \eta_k\rangle), \text{ and}\end{aligned}$
- for all $1 \leq i < h$ we have
  $\begin{aligned}\delta'(\langle q_k, i\rangle, \langle \sigma_k, \eta_k \cup \eta_l\rangle) &= \textstyle\bigwedge_{(c,s) \in (\eta_k(q_k) \cup \eta_l(q_k)) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \\ &= \textstyle\bigwedge_{(c,s) \in (\eta_k(q_k) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \wedge \\ &\qquad \textstyle\bigwedge_{(c,s) \in (\eta_l(q_k) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \\ &= \textstyle\bigwedge_{(c,s) \in (\eta_k(q_k) \setminus (D \times F_{2i}))} (c, \langle s, i\rangle) \\ &= \textstyle\bigwedge_{(c,s) \in (\eta_k(q_k) \setminus (D \times (F_{2i} \cap Q_k)))} (c, \langle s, i\rangle) \\ &= \delta'_k(\langle q_k, i\rangle, \langle \sigma_k, \eta_k\rangle).\end{aligned}$

Thus we can reuse the transitions computed for a UCT A'_k (i.e., for $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$) when computing the transitions of A'_{k+1} for $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_{k+1}$. Similar theorem holds for the UCT to NBT step:

Theorem 3. *Let* $A''_k = \langle \Sigma'_k, D, Q''_k, q''_{i_k}, \delta''_k, \alpha''_k\rangle$ *be NBT translation of* A'_k*, and* $A'' = \langle \Sigma', D, Q'', q''_i, \delta'', \alpha''\rangle$ *be the NBT translation of* A'*.*

Then for all $\langle S, O, g\rangle \in Q''_k$*,* $\sigma' = \langle \sigma_k, \eta_k\rangle \in \Sigma'_k$*,* $\sigma = \langle \sigma_k, \eta_k \cup \eta_l\rangle \in \Sigma'$*,* $\delta''(\langle S, O, g \cup f\rangle, \sigma) = \delta''_k(\langle S, O, [g \cup f/g]\rangle, \sigma')$ *where* $g : Q'_k \to \{0, \ldots, k'\}$ *(*$k' = (2n_k!)n_k^{2n_k}3^{n_k}(n_k+1)/n_k!$ *where* n_k *is the number of states in* A'_k*), and* $f : Q' \setminus Q'_k \to \{0, \ldots, k\}$*.*

Proof. If $O \neq \emptyset$ then

$$\begin{aligned}
&\delta''(\langle S, O, g \cup f\rangle, \sigma) \\
&\quad = \bigwedge_{c \in D} \bigvee_{\substack{g_c \, covers \, \langle g,\sigma,c\rangle \\ f_c \, covers \, \langle f,\sigma,c\rangle}} \langle \gamma'(S,\sigma,c), \gamma'(O,\sigma,c) \setminus odd(g_c \cup f_c), g_c \cup f_c\rangle \\
&\quad = \bigwedge_{c \in D} \bigvee_{\substack{g_c \;\; covers \;\; \langle g,\sigma,c\rangle \\ f_c \;\; covers \;\; \langle f,\sigma,c\rangle}} \langle \gamma'_k(S,\sigma',c), \gamma'_k(O,\sigma',c) \setminus odd(g_c), g_c \cup f_c\rangle \\
&\quad = \delta''_k(\langle S, O, [g \cup f/g]\rangle, \sigma')
\end{aligned}$$

The proof works analogously for the $O = \emptyset$ case.

This result shows that we can reuse the transitions we compute for an NBT A''_k used for checking the satisfiability of $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_k$ when we are computing the transitions of A''_{k+1} for $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_{k+1}$.

Example 1. Consider a formula $\varphi = \varphi_1 \wedge \varphi_2$ such that $\varphi_1 = \nu X.(\psi \wedge \langle - \rangle X)$ $\varphi_2 = \neg \nu X.(\psi \wedge \langle - \rangle X)$ where $\psi = \mu Y.(b \vee \langle - \rangle Y)$. Let $\Sigma = \{a, b\}$, and $D = \{1, 2\}$, an APT accepting models (which are tree models that have at least one path with infinitely many b's) of φ_1 is $A_1 = \{\Sigma, D, Q_1, q_1, \delta_1, \alpha_1\}$ where:

$Q_1 = \{q_0, q_1\}$
$\delta_1(q_0, a) = (1, q_0) \vee (2, q_0)$
$\delta_1(q_0, b) = (1, q_1) \vee (2, q_1)$
$\delta_1(q_1, a) = (1, q_0) \vee (2, q_0)$
$\delta_1(q_1, b) = (1, q_1) \vee (2, q_1)$
$\alpha_1 = \{\{q_0\}, \{q_0, q_1\}, \{q_0, q_1\}, \{q_0, q_1\}\}$

APT for φ_2, $A_2 = \{\Sigma, D, Q_2, q_2, \delta_2, \alpha_2\}$:

$Q_2 = \{q_2, q_3\}$
$\delta_2(q_2, a) = (1, q_2) \wedge (2, q_2)$
$\delta_2(q_2, b) = (1, q_3) \wedge (2, q_3)$
$\delta_2(q_3, a) = (1, q_2) \wedge (2, q_2)$
$\delta_2(q_3, b) = (1, q_3) \wedge (2, q_3)$
$\alpha_2 = \{\{\}, \{q_2\}, \{q_2, q_3\}, \{q_2, q_3\}\}$

and the APT for φ is $A_3 = \{\Sigma, D, Q_3, q_4, \delta_3, \alpha_3\}$:

$Q_3 = Q_1 \cup Q_2 \cup \{q_4\}$
$\delta_3 = \delta_1 \cup \delta_2$ plus the following transitions:
$\delta_3(q_4, a) = (1, q_0) \wedge (1, q_2)$
$\delta_3(q_4, b) = (1, q_0) \wedge (1, q_2)$
$\alpha_3 = \{\{q_0\}, \{q_0, q_1, q_2\}, \{q_0, q_1, q_2, q_3\}, \{q_0, q_1, q_2, q_3\}\}$

Note that the index of A_3 is 4 and A_1 and A_2 have the same index as A_3 according to the definition of α_k (for k=1 and k=2 in this case). As a result some sets in α_1 and α_2 are repeated at the end.

The incremental strategy used for this formula first checks for the emptiness of A_1 (which is not empty), then checks for the emptiness of A_3 (while re-using the transitions computed for the UCT A'_1 and the NBT A''_1), e.g.: $\delta'_3(\langle q_0, 0\rangle, \langle a, \eta_1 \cup \eta_2\rangle) = \delta'_1(\langle q_0, 0\rangle, \langle a, \eta_1\rangle)$ for all $\eta_1 \in R_1$ and $\eta_2 \in R_2$. Here R_1 is the set of restrictions $\eta_1 : Q_1 \to 2^{D \times Q_1}$ such that for all $\langle \sigma_1, \eta_1\rangle \in \Sigma'_1$, η_1 is relevant to σ_1, and R_2 is the set of restrictions $\eta_2 : Q_3 \setminus Q_1 \to 2^{D \times Q_3}$.

3.2 The Algorithm

Let A_i be the APT for $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_i$, and let A'_i be the UCT translation of A_i, $A''_i[j]$ be the NBT translation of A'_i where $\mathcal{R}$ is the set of functions $f : Q'_i \to \{0, \ldots, j\}$ for $1 \leq i \leq n$. The algorithm outlined in Figure 1 incrementally

```
 1: initial = 1
 2: for i = 1 to n do
 3:    construct A_i
 4:    if i > 1 then
 5:       construct A'_i using A'_{i-1}
 6:    end if
 7:    k = (2n!)n^{2n}3^n(n+1)/n! for A'_i with n states
 8:    for j = initial to k - 1 do
 9:       construct A''_i[j]
10:       if A''_i[j] is not empty then
11:          if i = n then
12:             return not empty
13:          else
14:             initial = j
15:             go to 2
16:          end if
17:       end if
18:    end for
19:    if A''_i[k] is empty then
20:       return empty
21:    end if
22: end for
```

Fig. 1. Pseudo-code for Incremental Satisfaction Algorithm

constructs automata $A''_i[j]$ representing φ_i for $1 \leq i \leq n$ and looks for the smallest j, $j_m \leq k$ such that $A''_i[j]$ is not empty reusing the automaton $A''_i[j]$ in the computation of $A''_i[j+1]$. If $A''_i[k]$ is empty it stops, if not it constructs $A''_{i+1}[j_m]$ reusing the automaton $A''_i[j_m]$. Hence we have two directions first we are checking for the emptiness of a particular automaton $A''_i[j]$ for $1 \leq j \leq k$, second we are checking for the emptiness of automata $A''_i[j]$ for $1 \leq i \leq n$. We

are using the proposed incremental technique on computing automata reusing the previous automata in both directions.

Theorem 4. *If $A''_i[k']$ is not empty then $A''_{i-1}[k']$ is also not empty where $1 \leq k' \leq (2n!)n^{2n}3^n(n+1)/n!$ and n is the number of states in A'_i.*

Proof. Let $A'_i = \langle \Sigma, D, Q, q_i, \delta, \alpha \rangle$, and $A'_{i-1} = \langle \Sigma_1, D, Q_1, q_{i_1}, \delta_1, \alpha_1 \rangle$ Starting state of A''_i is $q''_i = \langle \{\langle q_i, 0\rangle\}, \emptyset, g_0 \rangle$, and starting state of A''_{i-1} is $q''_{i_1} = \langle \{\langle q_{i_1}, 0\rangle\}, \emptyset, g_0^1 \rangle$ where R is the set of functions $f : Q \to \{0, \ldots k'\}$ and $g_0 \in R$ and $g_0^1 \in R$ map all the states in Q and Q_1 to k' respectively. For each $c \in D$, q''_i goes to $\langle S, O, g_0 \rangle$. If we remove all the states $Q_2 = Q \setminus Q_1$ from a path π that start with $\langle S, O, g_0 \rangle$ then we get a path π_2 that start with $\langle S_2, O_2, g_0^1 \rangle$ where $\langle q_{i_1}, 0\rangle \in S_2$ and $\langle q_{i_1}, 0\rangle \in O_2$. For each $c \in D$, q''_{i_1} goes to $\langle S_1, O_1, g_0^1 \rangle$ where $S_1 = \{\langle q_{i_1}, 0\rangle\}$ and $O_1 = \{\langle q_{i_1}, 0\rangle\}$. Let a path that start with $\langle S_1, O_1, g_0^1 \rangle$ be π_1. If π_2 is accepting then π_1 is also accepting since $O_1 \subseteq O_2$ and we get to $\emptyset$ from O_1 if we get to $\emptyset$ from O_2.

This theorem shows that the smallest j such that $A''_{i-1}[j]$ is not empty is also the smallest possible j such that $A''_i[j]$ is not empty. As a consequence, when we are constructing $A''_i[j]$ we can start from the *last* j. Also, this means we can directly reuse the information computed at stage $i-1$.

3.3 A Top-Down Approach to the APT to NBT Translation and to the NBT Emptiness Algorithm

We represent the general construction algorithm as a logic program and check the emptiness using a goal with respect to the program. The outline of the program for the construction of an NBT A''_n for a formula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ is as follows:

- $\mathsf{sat}(A''_1[1]) \leftarrow \mathsf{check}(A''_1[1], \_)$
- For all $1 < i \leq k$ and $1 < j \leq n$:
  $\mathsf{sat}(A''_j[i]) \leftarrow \mathsf{sat}(A''_j[i-1])$
  $\mathsf{sat}(A''_j[i]) \leftarrow \mathsf{exists}(A''_j[i-1]), \mathsf{check}(A''_j[i], A''_j[i-1])$
  $\mathsf{sat}(A''_j[i]) \leftarrow \mathsf{exists}(A''_{j-1}[i]), \mathsf{check}(A''_j[i], A''_{j-1}[i])$
  $\mathsf{exists}(A''_j[i]) \leftarrow \mathsf{check}(A''_j[i], \_)$

We ask the following query with respect to this program:

$$\mathsf{sat}(A''_1[k]), \mathsf{sat}(A''_2[k]), \ldots, \mathsf{sat}(A''_n[k])$$

Here, $\mathsf{sat}(A''_j[i])$ is true if $A''_j[i]$ is not empty. The predicate $\mathsf{check}(A''_j[i], A''_l[m])$ checks the emptiness of $A''_j[i]$ using $A''_l[m]$ and returns true if $A''_j[i]$ not empty. If an automaton $A''_j[i]$ is constructed $\mathsf{exists}(A''_j[i])$ is marked as true. As a result we check the emptiness of A''_1, A''_2, ... A''_n and stop if we hit an empty one using a top-down approach with memoing where the automata we compute are kept in the memo tables to be used whenever needed where the construction rules ensure that we reuse the automaton we compute at a particular stage in the next stage.

The proposed NBT emptiness algorithm for a particular automaton (implementation of the check predicate) checks if subtrees which have only final nodes in their leaves are repeated infinitely often. The emptiness query works top-down starting from the transitive closure of the initial state on these types of subtrees and stops checking when it makes certain that they are repeated infinitely often. This means that there is a tree accepted by the automaton. We compute only the transitions that we need to answer the emptiness query. For instance, to answer the emptiness query on an NBT automaton we only need to compute the transitions that are reachable from the starting state of the automaton.

Example 2. Consider an NBT automaton A where $\Sigma = \{a\}$, $D = \{1, 2\}$ $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $q_i = q_0$, $\delta(q_0, a) = (1, q_1) \wedge (2, q_2)$, $\delta(q_1, a) = (1, q_2) \wedge (2, q_3)$, $\delta(q_2, a) = (1, q_1) \wedge (2, q_1)$, $\delta(q_3, a) = (1, q_1) \wedge (2, q_3)$, $\delta(q_4, a) = (1, q_5) \wedge (2, q_5)$, $\delta(q_5, a) = (1, q_1) \wedge (2, q_1)$, and $\alpha = \{q_2, q_3\}$

When we are running the emptiness algorithm on this automaton we only compute the first four transitions.

4 Heuristics

In this section, we provide several heuristics and optimizations that can be applied to the proposed technique. First, we explain the optimizations in translation of an APT A to a UCT A' which is an incremental technique on the alphabet we use for A'. Then we explain the optimizations in translation of a UCT A' to an NBT A'' which is an incremental technique on the size of the functions in $\mathcal{R}$ we use for A'' which is proposed in [19]. Finally, we describe the heuristics we can use for rewriting conjunctive formulas (i.e. reordering the subformulas in a conjunctive formula) so that we have a better chance for detecting possible contradictions faster.

Optimizations in APT to UCT Translation. First we introduce an optimization used in the translation of APT to UCT.

Since $\Sigma' \subseteq \Sigma \times R$ we can start the construction using a subset Σ'_1 of Σ'. We proceed with a larger subset, Σ'_2, if the satisfiability query is empty, and repeat enlarging the alphabet until either the query becomes non-empty or we reach to the set Σ'. We are also able to reuse the results in the next computation since $\Sigma'_1 \subseteq \Sigma'_2$.

Theorem 5. *Let* $A'_1 = \langle \Sigma'_1, D, Q, q_i, \delta'_1, \alpha \rangle$ *and* $A'_2 = \langle \Sigma'_2, D, Q, q_i, \delta'_2, \alpha \rangle$ *are UCT translations of an APT* A *using* Σ'_1 *as alphabet of* A'_1 *and using* Σ'_2 *as alphabet of* A'_2. *If* $\Sigma'_1 \subseteq \Sigma'_2$, *then* $\delta'_1 \subseteq \delta'_2$.

Proof. Since we define $\delta'_2(\langle q, i \rangle, \langle \sigma_2, \eta_2 \rangle)$ for every $q \in Q$, $\sigma_2 \in \Sigma_2$, $\eta_2 \in R_2$, and for all $0 \leq i < h$ where R_2 is the set of restrictions such that for all $\langle \sigma_2, \eta_2 \rangle \in \Sigma'_2$, η_2 is relevant to σ_2 the same way as $\delta'_1(\langle q, i \rangle, \langle \sigma_1, \eta_1 \rangle)$ for every $q \in Q$, $\sigma_1 \in \Sigma_1$, $\eta_1 \in R_1$, and for all $0 \leq i < h$ where R_1 is the set of restrictions such that for all $\langle \sigma_1, \eta_1 \rangle \in \Sigma'_1$, η_1 is relevant to σ_1 then if $\Sigma'_1 \subseteq \Sigma'_2$, $\delta'_1 \subseteq \delta'_2$.

Optimizations in UCT to NBT Translation. In the proposed translation of UCT to NBT we start from an initial value k_1 for k and increase this value up to k_2, as long as the satisfiability query is empty. We continue this process until either the automaton becomes non-empty or we reach the upper bound of $(2n!)n^{2n}3^n(n+1)/n!$ for n the number of states in the UCT automaton. This approach has been proposed in [19]. Our decomposition, however, allows an incremental implementation that reuses *the transitions* computed for k_1 in the subsequent construction for k_2.

Theorem 6. *Let $A''_1[k_1]$ and $A''_2[k_2]$ are NBT translations of an APT A, using k_1 as the maximum range of functions in $\mathcal{R}_1$ for A''_1 and k_2 as the maximum range of functions in $\mathcal{R}_2$ for A''_2. If $k_1 \leq k_2$, then $\delta''_1 \subseteq \delta''_2$.*

Proof. Since $\mathcal{R}_1$ is the set of functions $f_1 : Q' \to \{0, \ldots, k_1\}$ and $\mathcal{R}_2$ is the set of functions $f_2 : Q' \to \{0, \ldots, k_2\}$ and $k_1 \leq k_2$ then $\mathcal{R}_1 \subseteq \mathcal{R}_2$ which means $Q''_1 \subseteq Q''_2$. Thus $\delta''_1 \subseteq \delta''_2$.

Example 3. Consider an alternating automaton A such that: $\Sigma = \{a\}$, $D = \{1, 2\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $q_i = q_0$, $\delta(q_0, a) = (1, q_1) \wedge (2, q_2)$, $\delta(q_1, a) = (1, q_3) \wedge (2, q_3)$, $\delta(q_2, a) = (1, q_3) \wedge (2, q_3)$, $\delta(q_3, a) = (1, q_3)$, and $\alpha = \{\{\}, \{q_0, q_1, q_2, q_3\}\}$ We have calculated the actual number of transitions in the UCT translation of A, A' and the NBT translation of A, A'', and the number of transitions we need to answer the satisfiability query after we apply the above optimizations. The set of restrictions is R and the set of restrictions we used for answering the satisfiability query is R_1. The number of transitions computed for A' with R is 4×2^{32} and the number of transitions computed for A' with R_1 is 4. The results for the NBT translation are given in Figure 2 where $k = 2^{20} \cdot 42525$, and $k_1 = 1$.

# of transitions computed for $A''[k]$	$256 \cdot k^4$
# of transitions computed for $A''[k_1]$	256
# of transitions computed for $A''[k_1]$ with top-down evaluation	70

Fig. 2. Number of transitions in the NBT automata $A''[k]$ and $A''[k_1]$

Heuristics for Ordering of Conjunctive Formulas. Consider a logical consequence question $\{\varphi_1, \varphi_2, \ldots, \varphi_n\} \models \psi$, such that the formula ψ is already inconsistent with a subset of formulas in $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$. As we use an incremental technique we can use rewriting heuristics to generate a formula $\neg\psi \wedge \varphi_{i_1} \wedge \varphi_{i_2} \wedge \ldots \wedge \varphi_{i_n}$ such that $[i_1, i_2, \ldots i_n]$ is a permutation of $[1, 2, \ldots n]$. For instance, the formulas $\varphi_1, \varphi_2, \ldots, \varphi_n$ can be ordered according to the number of free variables they share with ψ. Hence we improve our chances of finding a possible contradiction faster if we use this formula instead of the original one in the proposed algorithm. The following examples demonstrate the effect of ordering of the subformulas of a conjunctive formula.

Example 4. Consider a formula $\psi = \varphi \wedge \varphi_4$ where φ is the formula given in Example 1, $\varphi_4 = \nu X.(\psi \wedge \langle - \rangle X)$ such that $\psi = \mu Y.(a \vee \langle - \rangle Y)$, $\Sigma = \{a, b\}$, and $D = \{1, 2\}$, an APT for φ_4 is $A_4 = \{\Sigma, D, Q_4, q_5, \delta_4, \alpha_4\}$ where:

$Q_4 = \{q_5, q_6\}$
$\delta_4(q_5, a) = (1, q_5) \vee (2, q_5)$
$\delta_4(q_5, b) = (1, q_6) \vee (2, q_6)$
$\delta_4(q_6, a) = (1, q_5) \vee (2, q_5)$
$\delta_4(q_6, b) = (1, q_6) \vee (2, q_6)$
$\alpha_4 = \{\{q_6\}, \{q_5, q_6\}, \{q_5, q_6\}, \{q_5, q_6\}\}$

APT for ψ, $A_5 = \{D, \Sigma, Q_5, q_7, \delta_5, \alpha_5\}$:

$Q_5 = Q_3 \cup Q_4 \cup \{q_7\}$
$\delta_5 = \delta_3 \cup \delta_4$ plus the following transitions:
$\delta_5(q_7, a) = (1, q_4) \wedge (1, q_5)$
$\delta_5(q_7, b) = (1, q_4) \wedge (1, q_5)$
$\alpha_5 = \{\{q_0, q_6\}, \{q_0, q_1, q_2, q_5, q_6\}, \{q_0, q_1, q_2, q_3, q_5, q_6\}, \{q_0, q_1, q_2, q_3, q_5, q_6\}\}$

Using the proposed strategy we first check whether A_1 defined in Example 1 is empty (it is not empty), then we check the emptiness of A_3 which is empty and thus we do not need to construct A'_5 and A''_5. The estimated number of transitions is 10×2^{50} for A'_3, and 16×2^{128} for A'_5. The estimated number of transitions for A''_3 and A''_5 are given in Figure 3 where $k_3 = 20! \cdot 10^{20} \cdot 3^{10} \cdot 11/10!$, $k_5 = 32! \cdot 2^{128} \cdot 3^{16} \cdot 17/16!$.

estimated # of transitions for A''_3	$2 \times 2^{10} \times 2^{10} \times k_3^{10}$
estimated # of transitions for A''_5	$2 \times 2^{16} \times 2^{16} \times k_5^{16}$

Fig. 3. Number of transitions in the NBT automata A''_3 and A''_5

Example 5. Consider a logical consequence problem $\{\varphi_2, \varphi_3, \varphi_4, \varphi_5\} \models \varphi_1$ where φ_1 and φ_2 are given in Example 1, $\varphi_3 = \nu X.(\psi_1 \wedge \langle - \rangle X)$ such that $\psi_1 = \mu Y.(a \vee \langle - \rangle Y)$, $\Sigma_3 = \{a, b\}$, $\varphi_4 = \nu X.(\psi_2 \wedge \langle - \rangle X)$ such that $\psi_2 = \mu Y.(c \vee \langle - \rangle Y)$, $\Sigma_4 = \{c, b\}$, $\varphi_5 = \nu X.(\psi_3 \wedge \langle - \rangle X)$ such that $\psi_3 = \mu Y.(d \vee \langle - \rangle Y)$, $\Sigma_5 = \{d, b\}$, and $D = \{1, 2\}$, an APT for φ_3 is $A_3 = \{\Sigma, D, Q_3, q_5, \delta_3, \alpha_3\}$ where:

$Q_3 = \{q_5, q_6\}$
$\delta_3(q_5, a) = (1, q_5) \vee (2, q_5)$
$\delta_3(q_5, b) = (1, q_6) \vee (2, q_6)$
$\delta_3(q_6, a) = (1, q_5) \vee (2, q_5)$
$\delta_3(q_6, b) = (1, q_6) \vee (2, q_6)$
$\alpha_3 = \{\{q_6\}, \{q_5, q_6\}, \{q_5, q_6\}, \{q_5, q_6\}\}$

The APT A_4 for φ_4 and the APT A_5 for φ_5 are the same as A_3 except that the state names are changed and the letter a is replaced with c in A_4 and d in A_5, respectively.

Using the proposed strategy we first check if A_1 defined in Example 1 is empty (it is not empty), then we check the emptiness of the intersection automaton $A_{1,2}$ of A_1 and A_2 which is empty. Hence, we do not need to construct the complete intersection automaton A for A_1, A_2, A_3, A_4, and A_5. The estimated number of transitions for $A''_{1,2}$ and A'' are given in Figure 4 where $k_1 = 20! \cdot 10^{20} \cdot 3^{10} \cdot 11/10!$, $k_2 = 56! \cdot 28^{56} \cdot 3^{28} \cdot 29/28!$.

estimated # of transitions for $A''_{1,2}$	$2 \times 2^{10} \times 2^{10} \times k_1^{10}$
estimated # of transitions for A''	$5 \times 2^{28} \times 2^{28} \times k_2^{28}$

Fig. 4. Number of transitions in the NBT automata $A''_{1,2}$ and A''

5 Related Work

The connection between logic and automata was first considered by Büchi [5] and Elgot [10]. They have shown that monadic second-order logic over finite words and finite automata have the same expressive power, and we can transform formulas of this logic to finite automata and vice versa. Later, Büchi [6], McNaughton [18], and Rabin [21] proved that monadic second-order logic over infinite words (and trees) and finite automata also have the same expressive power. The practical use of this connection was investigated for temporal logics and fixed-point logics which led to the theory of model checking [3,28]. In addition, μ-calculus formulas can be translated to alternating automata [9,13,27]. Unfortunately, the standard way of checking for emptiness of an alternating automaton involves Safra's construction [22]. An alternative approach to this problem is proposed by Vardi and Kupferman [19] that does not use Safra's theorem. An extensive survey on automata and logic can be found in [25].

The connection between logics and automata theory has been used for implementing decision procedures for numerous logics, for example the MONA system [11,14] for deciding monadic second order logics on finite words and trees. It is argued that the success of these procedures relies on efficient operations on a compact representation of automata based on BDDs [14,15]. Recently, an extension of Safraless decision algorithm that is amenable to implementation was proposed for LTL formulas [20] which also improved the complexity of the algorithm.

6 Conclusions and Future Work

In this paper, we have developed an incremental approach to an automata-based decision procedure for μ-calculus. The proposed technique and optimizations are sufficiently general to be applicable to other automata-based techniques. Future research will follow several directions:

1. we attempt to reduce the part of the automaton needed to show satisfiability/unsatisfiability by introducing additional heuristics in the incremental construction,
2. for particular classes of problems, for which other techniques exhibit better performance due to reduced search space, we attempt to modify the proposed incremental approach to mimic those approaches, and
3. we study how the proposed incremental technique can take advantage of the structure of problems formulated in more restricted formalisms such as description logics.

References

1. Baader, F., Sattler, U.: An Overview of Tableau Algorithms for Description Logics. Studia Logica 69, 5–40 (2001)
2. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML Class Diagrams using Description Logic Based Systems. In: Baader, F., Brewka, G., Eiter, T. (eds.) KI 2001. LNCS (LNAI), vol. 2174, Springer, Heidelberg (2001)
3. Bernholtz, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 142–155. Springer, Heidelberg (1994)
4. Bradfield, J., Stirling, C.: Modal Mu-Calculi (chapter 12) (2006)
5. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik Grundl. Math. 6, 66–92 (1960)
6. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Proc. Int. Congr. for Logic, Methodology and Philosophy of Science 1–11 (1962)
7. Calvanese, D., Lenzerini, M., Nardi, D.: Description logics for conceptual data modeling. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, pp. 229–264. Kluwer, Dordrecht (1998)
8. Demri, S., Sattler, U.: Automata-theoretic decision procedures for information logics. Fundam. Inform. 53(1), 1–22 (2002)
9. Jutla, C.S., Emerson, E.A.: Tree automata, mu-calculus and determinacy. In: Proceedings of the 32nd annual symposium on Foundations of computer science, pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)
10. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Amer. Math. Soc. 98, 21–52 (1961)
11. Henriksen, J.G., Jensen, J.L., Jörgensen, M.E., Klarlund, N., Paige, R., Rauhe, T., Sandholm, A.: MONA: Monadic second-order logic in practice. In: Brinksma, E., Steffen, B., Cleaveland, W.R., Larsen, K.G., Margaria, T. (eds.) TACAS 1995. LNCS, vol. 1019, pp. 89–110. Springer, Heidelberg (1995)
12. Hladik, J., Sattler, U.: A Translation of Looping Alternating Automata into Description Logics. In: Baader, F. (ed.) Automated Deduction – CADE-19. LNCS (LNAI), vol. 2741, pp. 90–105. Springer, Heidelberg (2003)
13. Janin, D., Walukiewicz, I.: Automata for the modal μ-calculus and related results. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 552–562. Springer, Heidelberg (1995)
14. Klarlund, N.: MONA & FIDO: The logic-automaton connection in practice. Computer Science Logic 311–326 (1997)
15. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. Int. J. Found. Comput. Sci. 13(4), 571–586 (2002)

16. Kozen, D.: Results on the propositional *mu*-calculus. Theoretical Computer Science 27, 333–354 (1983)
17. Wolper, P., Vardi, M.Y.: Automata theoretic techniques for modal logics of programs (extended abstract). In: STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing, pp. 446–456. ACM Press, New York (1984)
18. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Information and Control 9, 521–530 (1966)
19. Vardi, M.Y., Kupferman, O.: Safraless decision procedures. In: Proc. 46th IEEE Symp. on Foundations of Computer Science, pp. 531–540, Pittsburgh (October 2005)
20. Vardi, M.Y., Kupferman, O., Piterman, N.: Safraless compositional synthesis. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 31–44. Springer, Heidelberg (2006)
21. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35 (1969)
22. Safra, S.: On the Complexity of ω-Automata. In: FOCS, pp. 319–327 (1988)
23. Safra, S.: Exponential Determinization for omega-Automata with Strong-Fairness Acceptance Condition (Extended Abstract). In: STOC, pp. 275–282 (1992)
24. Sattler, U., Vardi, M.Y.: The Hybrid μ-Calculus. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 76–91. Springer, Heidelberg (2001)
25. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, vol. 3, Springer, Heidelberg (1997)
26. Vardi, M.Y.: What makes Modal Logic so Robustly Decidable. Descriptive Complexity and Finite Models. American Mathematical Society (1997)
27. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
28. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. LICS, pp. 322–331 (1986)