Synthesis of Uninitialized Systems*

Thomas A. Henzinger[1], Sriram C. Krishnan[2],
Orna Kupferman[3], Freddy Y.C. Mang[4]

[1]Electrical Engineering and Computer Sciences, University of California at Berkeley.
Email: tah@eecs.berkeley.edu
[2]Cisco Systems, Inc.
Email: srikrish@cisco.com
[3]School of Computer Science and Engineering, Hebrew University.
Email: orna@cs.huji.ac.il
[4]Advanced Technology Group, Synopsys, Inc.
Email: fmang@synopsys.com

Abstract. The sequential synthesis problem, which is closely related to Church's solvability problem, asks, given a specification in the form of a binary relation between input and output streams, for the construction of a finite-state stream transducer that converts inputs to appropriate outputs. For efficiency reasons, practical sequential hardware is often designed to operate without prior initialization. Such hardware designs can be modeled by uninitialized state machines, which are required to satisfy their specification if started from any state. In this paper we solve the sequential synthesis problem for uninitialized systems, that is, we construct uninitialized finite-state stream transducers. We consider specifications given by LTL formulas, deterministic, nondeterministic, universal, and alternating Büchi automata. We solve this *uninitialized synthesis problem* by reducing it to the well-understood initialized synthesis problem. While our solution is straightforward, it leads, for some specification formalisms, to upper bounds that are exponentially worse than the complexity of the corresponding initialized problems. However, we prove lower bounds to show that our simple solutions are optimal for all considered specification formalisms. We also study the problem of deciding whether a given specification is uninitialized, that is, if its uninitialized and initialized synthesis problems coincide. We show that this problem has, for each specification formalism, the same complexity as the equivalence problem.

1 Introduction

In *sequential synthesis*, we transform a temporal specification into a reactive system that is guaranteed to satisfy the specification. A *closed system* that meets the specification can be extracted from a model that satisfies the specification, that is, the synthesis of closed systems amounts to solving a satisfiability ($\exists$) problem [EC82]. However, as argued for transformational systems in [MW80], and for

* This research was supported in part by the SRC contract 99-TJ-683.003, the DARPA contract NAG2-1214, and the NSF grant CCR-9988172.

reactive systems in [ALW89,Dil89,PR89], the synthesis of *open systems*, which interact with an unknown environment, requires the solution of a $\forall\exists$ problem: for all sequences of inputs, there exists a sequence of outputs that satisfies the specification. Consider, for example, a scheduler for a printer that serves two users. The scheduler is an open system. Each time unit it reads the input signals $J1$ and $J2$ (a job sent from the first or second user, respectively), and writes the output signals $P1$ and $P2$ (print a job of the first or second user, respectively). The scheduler should be designed so that jobs of the two users are not printed simultaneously, and whenever a user sends a job, the job is printed eventually. Of course, this should hold no matter how the users send jobs. We can specify the requirement for the scheduler in terms of a *linear temporal logic* (LTL) formula ψ [Pnu81], such as

$$\Box(J1 \Rightarrow \bigcirc(\neg J1\, \mathcal{U}\, P1)) \wedge \Box(J2 \Rightarrow \bigcirc(\neg J2\, \mathcal{U}\, P2)) \wedge \Box\neg(J1 \wedge J2).$$

Evidence of ψ's satisfiablity (note that ψ is satisfied in a structure in which the four signals never occur) is not of much help in extracting a correct scheduler: while such evidence only suggests a scheduler that is guaranteed to satisfy ψ for *some* input sequence, we want a scheduler that satisfies ψ for *all* possible scripts of jobs sent to the printer.

We now make this intuition formal. A *stream transducer* is a function that, given an infinite sequence of inputs, produces an infinite sequence of outputs. In particular, for the set I of inputs signals and the set O of output signals, a stream transducer is a function from $(2^I)^\omega$ to $(2^O)^\omega$. A *stream requirement* is a binary relation between input streams and output streams; that is, a stream requirement is a subset of $(2^I)^\omega \times (2^O)^\omega$ or, equivalently, a set of infinite words in $(2^{I\cup O})^\omega$. The stream transducer T *realizes* the stream requirement R if for every input stream $\tau \in (2^I)^\omega$, we have $R(\tau, T(\tau))$. Stream requirements can be specified by LTL formulas over the set $I \cup O$ of atomic propositions, or by *automata on infinite words* over the alphabet $2^{I\cup O}$. Stream transducers can be implemented by state machines that proceed ad infinitum. The *finite-state implementation* of a stream transducer is a deterministic finite-state machine that, from a given state on a given set of input signals, generates a set of output signals and moves to a successor state. The *realizability problem* (RP) asks, given a stream requirement R, if there is a finite-state implementation of a stream transducer that realizes R. The *sequential synthesis problem*, then, is to find a finite-state implementation (if one exists). The RP was first stated by Church [Chu62] for stream requirements specified in the *sequential calculus*. Since then, several solutions for the RP have been studied: [BL69,Rab72] showed that the RP is quadratic (exponential) if the specification is a deterministic (nondeterministic) Büchi automaton; [PR89] showed that the RP is doubly exponential if the specification is an LTL formula (researchers from control theory also studied the RP in the context of supervisory control for discrete-event systems [RW89]). The solutions to the RP can be extended, within the same complexity bounds, to construct finite-state implementations, so that a solution to the RP immediately provides a solution also to the sequential synthesis problem [Rab70,MS95,KV99].

In practice, sequential hardware is often designed to operate without prior initialization; that is, it is supposed to satisfy its input-output requirements if

started from any state. *Uninitialized state machines*, which model such hardware designs, require no reset circuitry and therefore have an advantage of smaller area. A well-known example of an uninitialized state machine is the IEEE 1149.1 standard for boundary-scan test [IEEE93]. Uninitialized state machines are also necessary for the *safe replaceability* of sequential circuits [SP94], where a state machine is replaced by another one in such a way that the surrounding environment is unable to detect the changes. The replacing state machine may power-up in an arbitrary state, and is therefore uninitialized. The verification problem of deciding whether an uninitialized state machine safely replaces another machine, is studied in [SP94]. The optimization problem for uninitialized state machines is studied in [QBSP96]. In this paper, we study the synthesis problem for uninitialized state machines.

Given a stream requirement R, the *uninitialized realizability problem* (URP) asks if there is a finite-state implementation M that realizes R no matter what the initial state of M is. The *uninitialized synthesis problem*, then, is to find such an M (if one exists). We study the URP for stream requirements that are specified by LTL formulas or Büchi automata. We consider deterministic and nondeterministic Büchi automata, as well as universal Büchi automata, which accept a word iff all runs are accepting, and alternating Büchi automata, which allow both nondeterministic and universal branching modes. The solution of the URP is quite straightforward, and is done by a reduction to the RP: if the stream requirement R is specified by an LTL formula, then the URP for R can be reduced to the RP for the LTL formula $\mathit{always}(R) = \Box R$; if R is specified by a Büchi automaton, then the URP for R can be reduced to the RP for the automaton $\mathit{always}(R)$, which is obtained from R by adding a universal self-loop at the initial state. It is not hard to see that an infinite word $w \in (2^{I \cup O})^\omega$ satisfies the specification $\mathit{always}(R)$ iff w and all its suffixes satisfy R. This implies that R is realizable by an uninitialized implementation iff $\mathit{always}(R)$ is realizable. As in the initialized case, a solution to the uninitialized synthesis problem follows immediately from a solution to the URP.

While the above solution is straightforward, it may lead to upper bounds that are exponentially worse than the complexity of the RP for the corresponding specification formalism. For example, while for LTL specifications both RP and URP are doubly exponential, for deterministic Büchi automata, where the RP is quadratic, the presented solution of the URP is exponential. The reason is that the automaton $\mathit{always}(R)$ has a universal branching mode, which R may not have, and this makes the URP exponentially harder. In particular, if R is a deterministic automaton, then $\mathit{always}(R)$ is universal, and if R is nondeterministic, then $\mathit{always}(R)$ is alternating. Can the exponential blow-up be avoided by a more sophisticated solution? We answer this question in the negative by proving corresponding lower bounds for the URP of all discussed formalisms. Unlike the upper bounds, the lower-bound proofs are not immediate, and are the main technical contributions of this paper. Our results imply that specification formalisms that support an easy implementation of the *always* operator, such as LTL and alternating automata, have, unlike deterministic and nondeterministic automata, already "built-in" the complexity of uninitialized synthesis.

We say that a stream requirement R is *uninitialized* if it is suffix-closed; that is, for all infinite words $w \in (2^{I \cup O})^\omega$, if $w \in R$, then $w' \in R$ for all suffixes w' of w. For example, the LTL specification $\Box p$ for an output signal p is uninitialized, as w satisfies $\Box p$ iff all suffixes of w satisfy $\Box p$. The *uninitialized specification problem* (USP) asks if a given stream requirement R is uninitialized. This is the same as asking if the two formulas, or automata, that specify R and $always(R)$ are equivalent. In the final section, we show that the USP has, for all considered specification formalisms, the same complexity as the equivalence problem. For uninitialized stream requirements, the URP coincides with the RP. As the equivalence problem is easier than the corresponding URP in all cases, it follows that for specification formalisms whose URP is harder than RP, there is an advantage to first checking if the specification is uninitialized.

2 Preliminary Definitions

Trees. Given a finite set D of directions, a D-tree is a set $T \subseteq D^*$ such that if $x \cdot d \in T$, where $x \in D^*$ and $d \in D$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot d \in T$, for $d \in D$, are the *successors* of x. Each node $x \in T$ has a direction $dir(x)$ in D, namely, $dir(\epsilon) = d^0$ for some designated $d^0 \in D$, and $dir(x \cdot d) = d$. A path π of the tree T is a set $\pi \subseteq T$ such that $\epsilon \in \pi$, and for every $x \in \pi$, exactly one successor of x is in π. Given two finite sets D and Σ, a Σ-labeled D-tree is a pair (T, V), where T is a D-tree, and $V : T \to \Sigma$ maps each node of T to a letter in Σ. We extend V to paths in the straightforward way: for a path $\pi = \{\epsilon, w_0, w_0 w_1, \ldots\}$, we have $V(\pi) = V(\epsilon)V(w_0)V(w_0 w_1)\ldots$ We say that a $(D \times \Sigma)$-labeled D-tree (T, V) is *D-exhaustive* if $T = D^*$, and for every node $w \in D^*$, we have $V(w) = (dir(w), \sigma)$ for some $\sigma \in \Sigma$.

Alternating Büchi automata. For a given finite set X, let $\mathcal{B}^+(X)$ be the set of positive boolean formulas over X. A subset $Y \subseteq X$ satisfies a formula $\theta \in \mathcal{B}^+(X)$ if the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \setminus Y$ satisfies θ. An *alternating Büchi automaton* $\mathcal{U} = (\Sigma, U, u_0, \delta, F)$ consists of a finite alphabet Σ, a finite set U of states, an initial state $u_0 \in U$, a transition function $\delta : U \times \Sigma \to \mathcal{B}^+(U)$, and a set $F \subseteq U$ of accepting states. The automaton $\mathcal{U}$ is *universal* if $\delta(u, \sigma)$, for all $u \in U$ and $\sigma \in \Sigma$, is a conjunction of states from U; *nondeterministic*, if $\delta(u, \sigma)$ is a disjunction of states from U; and *deterministic*, if $\delta(u, \sigma)$ is a single state from U. A *run* of $\mathcal{U}$ on an infinite word $w = w_0 w_1 \ldots$ in Σ^ω is an infinite U-labeled D-tree (T, r), where $D = \{1, \ldots, |U|\}$, such that $r(\epsilon) = u_0$ and the following holds: for all nodes $x \in T$, if $|x| = i$ and $r(x) = u$ and $\delta(u, w_i) = \theta$, then x has k successors $x_1, \ldots, x_k$, for some $k \leq |U|$, and $\{r(x_1), \ldots, r(x_k)\}$ satisfies θ. A run (T, r) is *accepting* if every infinite path of (T, r) visits the accepting set F infinitely often. An infinite word w is accepted by $\mathcal{U}$ if there exists a run (T, r) on w such that (T, r) is accepting. The *language* $L(\mathcal{U})$ is the set of infinite words accepted by $\mathcal{U}$.

Finite-state machines. A *finite-state machine* (FSM) $M = (I, O, Q, q_{in}, \rho, \lambda)$ consists of a finite set I of input signals, a finite set O of output signals, a finite state set Q, an initial state $q_{in} \in Q$, a transition function $\rho : Q \times 2^I \to Q$, and

an output function $\lambda : Q \to 2^O$. We assume that there is a special output signal *init* such that $init \in \lambda(q)$ iff $q = q_{in}$. We also assume that there is a nonempty set $In(q_{in}) \subseteq 2^I$ such that for each $i_0 \in In(q_{in})$, there is a $q \in Q$ such that $\rho(q, i_0) = q_{in}$; that is, the state q_{in} is reachable via some input (if this is not the case, we can add a new state from which q_{in} is reachable). An FSM M interacts with its environment through its input and output signals. Initially, M is at the initial state $q_0 = q_{in}$. The environment initiates the interaction by inputting some $i_0 \in In(q_{in})$. Then, M starts operating by outputting $\lambda(q_0)$, to which the the environment replies with some input $i_1 \in 2^I$. The FSM M replies by moving to the state $q_1 = \rho(q_0, i_1)$ and outputting $\lambda(q_1)$. Interaction then continues ad infinitum.

Hence, the FSM M can be viewed as a *strategy* $S_M : (2^I)^* \to 2^O$ that maps every finite sequence of inputs to an output. To define S_M formally, we first define the function $C_M : (2^I)^* \to Q$ that maps each finite input sequence to the state visited after the sequence has been read: $C_M(\epsilon) = q_{in}$, and $C_M(i_1 \dots i_n) = \rho(C_M(i_1 \dots i_{n-1}), i_n)$. The strategy S_M induced by M is then defined for every $w \in (2^I)^*$ by $S_M(w) = \lambda(C_M(w))$. Note that the first input i_0 merely initiates the interaction and does not have any effect on the behavior of M; it is disregarded in the definition of C_M. Each infinite sequence $i_0 i_1 \dots \in In(q_{in}) \cdot (2^I)^\omega$ induces a *computation* $(i_0, S_M(\epsilon))(i_1, S_M(i_1))(i_2, S_M(i_1 i_2)) \dots \in (2^I \times 2^O)^\omega$ of M. The *language* $L(M)$ is the set of all computations of M. We refer to the language also as a set of infinite words in $(2^{I \cup O})^\omega$, where $i_0 i_1 \dots$ induces the computation $(i_0 \cup S_M(\epsilon)) \cdot (i_1 \cup S_M(i_1)) \cdot (i_2 \cup S_M(i_1 i_2)) \dots \in (2^{I \cup O})^\omega$. The strategy S_M induces, for a given first input $i_0 \in In(q_{in})$, a *computation tree* whose branches correspond to external nondeterminism caused by different inputs, namely, the 2^I-exhaustive $(2^I \times 2^O)$-labeled 2^I-tree $((2^I)^*, V)$ such that each node $w \in (2^I \times 2^O)^*$ is labeled by $V(w) = (dir(w), S_M(w))$, where $dir(\epsilon) = i_0$. Note that all computation trees of M differ only in the first input.

3 The Uninitialized Realizability Problem

In this section we define and solve the uninitialized realizability problem. We first start with the (initialized) realizability problem. Given a specification R over the input signals I and output signals O, the *realizability problem* (RP) for R asks if there is an FSM M such that for all words $w \in L(M)$, we have $w \models R$. If so, we say that R is *realizable* by M. The *specification* R can be an LTL formula, or an alternating Büchi automaton. If R is an LTL formula, then the atomic propositions of R are $I \cup O$, and the relation $\models$ is the usual satisfaction relation. If R is an automaton, then the alphabet of R is $2^I \times 2^O$, and the relation $\models$ is the language membership relation, that is, $w \models R$ iff $w \in L(R)$. The realizability problem is closely related to *Church's solvability problem* [Chu62], and it has been shown that the problem is solvable in quadratic (exponential) time if R is a deterministic (nondeterministic) Büchi automaton [BL69,Rab72,Saf88,PR89], and in doubly exponential time if R is an LTL formula [PR89,KV99].

An *uninitialized FSM* $M = (I, O, Q, \rho, \lambda)$ is similar to an FSM except that there is no initial state. The language $L(M) = \bigcup_{q \in Q} L(M_q)$ of M is simply the

union of the languages $L(M_q)$, where $M_q = (I, O, Q, q, \rho, \lambda)$ is the FSM obtained from M by regarding the state $q \in Q$ as the initial state. Given a specification R over the input signals I and output signals O, the *uninitialized realizability problem* (URP) for R asks if there is an uninitialized FSM M such that $w \models R$ for all words $w \in L(M)$. If the answer is yes, we say that R is *uninitialized realizable* by M.

3.1 Reducing URP to RP

We solve the URP by reducing it to the RP. For that, we define, given a specification R over the input signals I and output signals O, the specification $always(R)$ over I and O such that, for all words $w \in (2^I \times 2^O)^\omega$, we have w satisfies $always(R)$ iff all suffixes w' of w satisfy R. It is not hard to see that the URP for R can be reduced to the RP for $always(R)$, as stated in the following theorem.

Theorem 1. *Let I and O be finite sets of input and output signals, respectively, and let R be a specification over I and O. Then R is uninitialized realizable iff $always(R)$ is realizable.*

Given a specification R, we construct the specification $always(R)$ as follows. First, if R is an LTL formula, it is not hard to see that $always(R) = \Box R$. Now, if R is an alternating Büchi automaton $\mathcal{U} = (\Sigma, U, u_0, \delta, F)$, then $always(\mathcal{U}) = (\Sigma, U \cup \{u_0'\}, u_0', \delta', F \cup \{u_0'\})$, where u_0' is a new state, and for all $\sigma \in \Sigma$, we have $\delta'(u_0', \sigma) = \delta(u_0, \sigma) \wedge u_0'$; and for all $u \in U$, we have $\delta'(u, \sigma) = \delta(u, \sigma)$. Intuitively, the automaton $always(\mathcal{U})$ behaves like $\mathcal{U}$ except that $always(\mathcal{U})$ always sends a copy of itself to the suffix of the input word whenever it makes a transition. It follows that for every word w, not only w has to be accepted by $\mathcal{U}$, but so do all its suffixes. Note that one copy of $always(\mathcal{U})$ keeps visiting u_0' forever, which is why we have to duplicate the original initial state. Formally, we have the following.

Proposition 1. *Let $\mathcal{U} = (\Sigma, U, u_0, \delta, F)$ be an alternating Büchi automaton. For all words $w \in \Sigma^\omega$, the automaton $always(\mathcal{U})$ accepts w iff $\mathcal{U}$ accepts all suffixes of w.*

3.2 URP Complexity

We can solve the URP for R by solving the RP for $always(R)$. The complexity of this simple solution depends on the type of the specification $always(R)$. In Table 1 below we describe the type of $always(R)$ given the type of R. It follows that if R is

R	$always(R)$
an LTL formula	an LTL formula
a deterministic or universal Büchi automaton	a universal Büchi automaton
a nondeterministic or alternating Büchi automaton	an alternating Büchi automaton

Table 1. The cost of moving from R to $always(R)$.

a deterministic or a nondeterministic Büchi automaton, then the type of *always*(R) is richer than that of R, which in turn implies that the presented reduction from URP to RP incurs a cost. We now analyze the complexity of the URP and show that this cost is unavoidable, that is, the simple solution to the URP is optimal. First we consider LTL specifications.

Theorem 2. *The URP for LTL is 2EXPTIME-complete.*

Proof. The upper bound follows from the fact that the RP for LTL is in 2EXPTIME [PR89], and *always*(R), for R in LTL, is also in LTL. For the lower bound, we show that the URP is at least as hard as the RP, which is 2EXPTIME-hard [Ros92]. Indeed, the RP for an LTL formula φ can be reduced to the URP for $init \Rightarrow \varphi$. ∎

We now turn to the various types of Büchi automata. While the upper bounds are easy, the lower bounds require complicated generic reductions. To illustrate the proof ideas, we begin by considering the *closed* RP and URP, where the set I of input signals is empty. In this case, the behavior of the desired FSM is independent of the environment, and RP coincides with the satisfiability problem. In particular, for deterministic Büchi automata, the closed RP is NLOGSPACE-complete. We show that the transition to *uninitialized* FSMs makes the problem exponentially harder.

Proposition 2. *The closed URP for deterministic Büchi automata is PSPACE-complete.*

An automata-theoretic problem that is well-known to be PSPACE-complete is the universality problem for *nondeterministic* automata [HU79,Wol82]. On the other hand, the universality problem for *deterministic* automata is NLOGSPACE-complete. The standard PSPACE lower bound proof is by reduction from the membership problem for polynomial space Turing machines: given a polynomial-space Turing machine T, one constructs a nondeterministic Büchi automaton $\mathcal{U}$ such that $\mathcal{U}$ accepts invalid or rejecting computations of T. The closed URP also has some flavor of "universality." It comes from the requirement that all suffixes of an infinite word need to satisfy the specification. However, the lower-bound proof is different; we construct a *deterministic* Büchi automaton that accepts an infinite word w as well as all its suffixes iff w is a *valid* and *accepting* computation of T.

Proof of Proposition 2. Consider a deterministic Büchi automaton R. When $I = \emptyset$, the set 2^I is a singleton and the universal Büchi automaton *always*(R) is realizable iff its language is not empty. Since the latter can be checked in PSPACE [MH84,VW94], the upper bound follows.

For the lower bound, we do a reduction from the membership problem of a polynomial space Turing machine. Let $T = (\Gamma, Q, \mapsto, q_0, F_{acc}, F_{rej})$ be a polynomial space Turing machine, where Q is the set of states, $q_0 \in Q$ is the initial state, $F_{acc} \subseteq Q$ and $F_{rej} \subseteq Q$ are respectively accepting and rejecting states, and $\mapsto$: $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function. Assume T starts with the initial configuration, i.e., T at state q_0 with its reading head pointing at the leftmost cell of the empty tape. We also assume that once T reaches an accepting

configuration, i.e., $q \in F_{acc}$, it "cleans" the tape content and restarts from the initial configuration. The machine T accepts the empty tape iff T has an infinite computation visiting the initial and accepting configurations infinitely often.

Assume T uses $s(n)$ tape cells to process an input of length n. We encode a configuration of T by a string $\sharp\gamma_1\gamma_2 \dots (q,\gamma_i) \dots \gamma_{s(n)}$ in $(2^O)^*$, where subsets of the output signals O are selected to encode the the alphabets $\Gamma \cup (Q \times \Gamma) \cup \{\sharp\}$, i.e., $2^O = \Gamma \cup (Q \times \Gamma) \cup \{\sharp\}$. That is, a configuration starts with the letter $\sharp$, followed by a string of letters $\gamma_j \in \Gamma$, except for one in $Q \times \Gamma$. The meaning of the string is that γ_j is the letter on the j-th tape cell, while the letter (q,γ_i) indicates in addition that T is at state q with its reading head pointing at the i-th tape cell. Let $c = \sharp\sigma_1\sigma_2 \dots \sigma_{s(n)}$, and $c' = \sharp\sigma'_1\sigma'_2 \dots \sigma'_{s(n)}$ be two configurations. If c' is the successor of c, then we know by the transition function of T what σ'_i for $1 \leq i \leq s(n)$ should be. We let $next(\sigma_{i-1},\sigma_i,\sigma_{i+1})$ denote our expectations for σ'_i.

We define a deterministic Büchi automaton $\mathcal{U} = (2^O, U, u_0, \delta, F)$ such that $\mathcal{U}$ accepts an input word $w = w_0w_1 \dots$ iff w satisfies the following conditions: (1) The *next* relation of T is satisfied for the first three letters in w, i.e., $w_{s(n)+2} = next(w_0,w_1,w_2)$; and (2) the initial and the accepting configurations are eventually reached. It follows that $\mathcal{U}$ accepts an infinite word w as well as all its suffixes iff T has an infinite computation visiting the initial and accepting configuration infinitely often. Both conditions can be specified by a deterministic Büchi automaton of size polynomial in T.

Thus, if there exists a word w such that w and all its suffixes are accepted by $\mathcal{U}$, then there exists a suffix w' of w such that w' encodes an accepting run of T. On the other hand, if T has an accepting run, then it can be encoded as an infinite string $w \in \Sigma_0^*$ all of whose suffixes (including w itself) are accepted by $\mathcal{U}$. ∎

We now consider the general, *open* URP, where $I \neq \emptyset$. For deterministic Büchi automata, where the RP is quadratic, the URP is harder than both the RP and the closed URP.

Theorem 3. *The URP for deterministic or universal Büchi automata is EXPTIME-complete.*

Proof. Consider a deterministic or a universal Büchi automaton $\mathcal{U}$. The automaton $always(\mathcal{U})$ is a universal automaton, whose RP can be solved in EXPTIME [Rab72,MS95].

For the lower bound, we use the input signals in I in order to encode branches and extend the proof of Theorem 2 to apply to alternating Turing machines. Consider an alternating linear-space Turing machine $T = (\Gamma, Q_u, Q_e, \mapsto, q_0, F_{acc}, F_{rej})$, where the disjoint sets of states Q_u and Q_e are respectively the universal and existential states, while the disjoint sets of states $F_{acc} \subseteq Q_e$ and $F_{rej} \subseteq Q_e$ are respectively the accepting and rejecting states. Their union is denoted by Q. Our model of alternation prescribes that $\mapsto \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L,R\}$ has a binary branching degree. When a universal or an existential state of T branches into two states, we distinguish between the left and the right branches. Accordingly, we use $(q,a) \mapsto^l (q_l,b_l,\Delta_l)$ and $(q,a) \mapsto^r (q_r,b_r,\Delta_r)$ to indicate that when T is in state $q \in Q_u \cup Q_e$ reading input symbol a, it branches to the left with (q_l,b_l,Δ_l) and to

the right with (q_r, b_r, Δ_r). We also assume that once T reaches an accepting configuration, it "cleans" the tape content and restarts from the initial configuration (i.e., empty tape and initial state at the left end of the tape).

Assume T uses $s(n)$ cells in its working tape in order to process an input of length n. A configuration of T is encoded in a similar way to how a configuration of a polynomial space Turing machine is encoded, except that a configuration starts with either $\sharp_l$ or $\sharp_r$. The letter $\sharp \in \{\sharp_l, \sharp_r\}$ marks the beginning of a configuration; moreover, since T has an existential mode, i.e., when the state q of T is in Q_e, the letter $\sharp$ also indicates a guess (left or right) for the accepting successor. A computation of T can then be encoded by a computation tree whose branches describe sequences of configurations of T. Note that the computation tree is unique if we ignore the distinction between $\sharp_l$ and $\sharp_r$. A run of T is a pruning of a computation tree in which all the universal configurations have both successors and all the existential configurations $c = \sharp\gamma_1\gamma_2 \ldots (q, \gamma_i) \ldots \gamma_{s(n)}$ have only the left (resp. right) successor if $\sharp = \sharp_l$ (resp. $\sharp_r$). The run is accepting if all branches in the pruned tree visit the initial and accepting configuration infinitely often.

Given an alternating linear-space Turing machine T as above, we construct a deterministic Buchi word automaton $\mathcal{U}$ such that $\mathcal{U}$ is uninitialized realizable iff T has an accepting run on the empty tape (clearly, proving a lower bound for deterministic automata, implies a bound also for universal ones). The automaton $\mathcal{U}$ has input signals I such that the subsets of I encode the set $\{l, r\}$, i.e., $2^I = \{l, r\}$. It also has output signals O such that $2^O = \{\sharp_l, \sharp_r\} \cup \Gamma \cup (Q \times \Gamma)$. Let $c = \sharp\sigma_1\sigma_2 \ldots \sigma_{s(n)}$ and $c' = \sharp'\sigma'_1\sigma'_2 \ldots \sigma'_{s(n)}$ be two configurations, and let $(d_0, \sharp)(d_1, \sigma_1) \ldots (d_{s(n)}, \sigma_{s(n)})(d'_0, \sharp')(d'_1, \sigma'_1) \ldots (d'_{s(n)}, \sigma'_{s(n)})$ be a word in $(2^I \times 2^O)^*$. The letter d'_0 indicates the direction of c' with respect to c: if $d'_0 = l$, then c' is the left successor of c, and if $d'_0 = r$, then c' is the right successor of c. Note that the direction of c' is given by d'_0, not by the letter $\sharp$ or $\sharp'$; the letter $\sharp$ is only the guess that T makes at c if c is an existential configuration. If $\sharp = \sharp_l$ (resp. $\sharp_r$), then T guesses that the left (resp. right) successor of c leads to an accepting run: if the guess of T is different from the successor information given by the input, we say that there is a mismatch between the input and the guess of T at the configuration. That is, a mismatch happens at c with $\sharp = \sharp_r$ and $d'_0 = l$, as well as with $\sharp = \sharp_l$ and $d'_0 = r$. Recall that we require every path of the computation tree of T to be legal and accepting. On the other hand, since T is alternating, only the paths in the computation tree that are guessed in existential configurations need to be accepting. We use $\sharp_l$ and $\sharp_r$ in order to detect mismatches, where paths that contain a mismatch are considered accepting.

If the configuration c' is a successor of the configuration c, we know by the transition relation of T what the "next" relation is. Now we have two "next" relations, one for left branching and one for right branching. Let $next^l$ and $next^r$ be the "next" relations for the left branch and right branch respectively. The definition of $next^l$ (resp. $next^r$)is similar to that of the $next$ relation in the polynomial space Turing machine case, except that only the transition function $\mapsto^l$ (resp. $\mapsto^r$) is considered, the letter $\sharp$ is in $\{\sharp_l, \sharp_r\}$, and $next^l(\sigma_{s(n)}, \sharp, \sigma'_1) \in \{\sharp_l, \sharp_r\}$.

The automaton $\mathcal{U}$ can be constructed as follows. On input of a word $w = (d_0, \sigma_0)(d_1, \sigma_1) \ldots$, $\mathcal{U}$ checks the following:

1. The "next" transition relations of T are satisfied, i.e., $\sigma_{s(n)+2} = next^l(\sigma_0, \sigma_1, \sigma_2)$ if $d'_0 = l$, and $\sigma_{s(n)+2} = next^r(\sigma_0, \sigma_1, \sigma_2)$ if $d'_0 = r$; and
2. either of the following is true:
   (a) Eventually there is a mismatch in the direction specified by the input and T at an existential configuration, i.e., w contains the string $(d_0, \sharp_0) \ldots (d_j, (q, \gamma_j)) \ldots (d_{s(n)}, \sigma_{s(n)})(d'_0, \sharp'_0)$, where $q \in Q_e$ and either $\sharp_0 = \sharp_r$ and $d'_0 = l$, or $\sharp_0 = \sharp_l$ and $d'_0 = r$.
   (b) The initial configuration is eventually reached, and thereafter the accepting configuration is also eventually reached.

All the above conditions can be specified by a deterministic Büchi word automaton linear in the size of T.

Given a path w of a (2^I)-exhaustive $(2^I \times 2^O)$-labeled 2^I-tree $((2^I)^*, \tau)$, if $\mathcal{U}$ accepts w and all the suffixes of w, then by condition (2), w is a valid branch of the computation tree of T; moreover, by condition (3), if w is a branch guessed by T, then infinitely often the initial and accepting configurations are reached. Thus, $\mathcal{U}$ accepts all suffixes of all branches of $((2^I)^*, \tau)$ iff T has an accepting run. ∎

The RP for nondeterministic Büchi automata can be solved in exponential time, while the RP for alternating Büchi automata requires doubly exponential time. The following theorem shows that the URP for nondeterministic Büchi automata is exponentially harder than the RP, while for alternating Büchi automata, there is no additional cost.

Theorem 4. *The URP for nondeterministic or alternating Büchi automata is 2EXPTIME-complete.*

Proof. Consider a nondeterministic or an alternating Büchi automaton $\mathcal{U}$. The automaton $always(\mathcal{U})$ is an alternating automaton, whose RP can be solved in 2EXPTIME [Rab72,MS95]. The lower bound proof is similar to that for the URP for deterministic Büchi automata in Theorem 3, except that now we reduce from the membership problem of an alternating exponential space Turing machine. ∎

4 Uninitialized Specifications

For a specification R over the input signals I and output signals O, we say that R is *uninitialized* if for every infinite word $w \in (2^I \times 2^O)^\omega$, we have w satisfies R iff all suffixes of w satisfy R. It is easy to see that if R is uninitialized, then every FSM M that realizes R induces an uninitialized FSM M' (obtained from M by dropping its initial state) that uninitialized realizes R. Hence, a solution of the URP for R can be obtained from a solution of the RP for R. The *uninitialized specification problem* (USP) for a specification R asks whether or not R is uninitialized.

Solving the USP for the specification R amounts to checking if R is equivalent to $always(R)$. Clearly, $always(R)$ implies R, thus we only need to check whether R implies $always(R)$. For LTL formulas, this can be done by checking the validity of $R \Rightarrow always(R)$, and for alternating Büchi automata we need to solve the language-containment problem $L(R) \subseteq L(always(R))$. We show that this simple

approach, like the simple solution for URP, is also optimal. The lower bounds can be obtained by reductions from both the satisfiability and validity problems for the various specification formalisms (for an alternating Büchi automaton $\mathcal{U}$ with alphabet Σ, we say that $\mathcal{U}$ is *satisfiable* if $L(\mathcal{U}) \neq \emptyset$, and $\mathcal{U}$ is *valid* if $L(\mathcal{U}) = \Sigma^\omega$).

Lemma 1. *The USP for LTL, deterministic, nondeterministic, universal, or alternating Büchi automata, is (1) at least as hard as the corresponding satisfiability problem, and (2) at least as hard as the corresponding validity problem.*

We can now obtain complexity bounds for the USP. Our results are summarized in Table 2. All bounds in the table are tight.

Theorem 5. *The USP for LTL, universal, nondeterministic, or alternating Büchi automata is PSPACE-complete.*

Proof. For the upper bound, recall that R is uninitialized iff R implies $always(R)$. If the specification R is an LTL formula, then checking validity of $R \Rightarrow always(R)$ is in PSPACE [SC85]. If R is an alternating (or universal) automaton, we have to check the language-containment problem $L(R) \subseteq L(always(R))$. For that, we can first construct a nondeterministic Büchi automaton R_n such that the size of R_n is exponential in the size of R and $L(R_n) = L(R)$ [MH84], and we construct a nondeterministic Rabin automaton R_c such that the size of R_c is exponential in the size of $always(R)$ and R_c complements $always(R)$ (that is, $L(R_c) = \Sigma^\omega \setminus L(always(R))$.) [MS95]. Now, the product of R_n and R_c is a nondeterministic Rabin automaton whose emptiness can be checked in nondeterministic logarithmic space, implying a PSPACE upper bound for the USP.

The lower bound follows from Lemma 1, and from the fact that the satisfiability problem for LTL and universal (or alternating) Büchi automata, as well as the validity problem for nondeterministic Büchi automata are PSPACE-hard [SC85,HU79,Wol82]. ∎

Theorem 6. *The USP for deterministic Büchi automata is NLOGSPACE-complete.*

Proof. For a deterministic Büchi automaton R, the automaton $always(\mathcal{U})$ is universal, thus its complement R_c is a nondeterministic co-Büchi automaton. The product of R and R_c can be defined as a nondeterministic Rabin automaton, whose emptiness problem can be solved in nondeterministic logarithmic space, implying an NLOGSPACE upper bound for the USP.

The lower bound follows from Lemma 1, and from the fact that the satisfiability problem for deterministic Büchi automata is NLOGSPACE-hard. ∎

References

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th Intl. Colloquium on Automata, Languages, and Programming*, LNCS 372, pages 1–17. Springer-Verlag, 1989

	URP	RP	USP = Equivalence
LTL formulas	2EXPTIME	2EXPTIME	PSPACE
deterministic Büchi automata	EXPTIME	quadratic	NLOGSPACE
nondeterministic Büchi automata	2EXPTIME	EXPTIME	PSPACE
universal Büchi automata	EXPTIME	EXPTIME	PSPACE
alternating Büchi automata	2EXPTIME	2EXPTIME	PSPACE

Table 2. The complexity of the RP, URP, and USP.

[BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[Chu62] A. Church. Logic, arithmetic, and automata. In *Proc. Intl. Congress of Mathematicians*, pages 23–35. Institut Mittag-Leffler, 1962.

[Dil89] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits.* MIT Press, 1989.

[EC82] E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1987.

[IEEE93] IEEE Standard 1149.1-1993. *IEEE Standard Test Access Port and Boundary Scan Architecture.* IEEE, 1993.

[KV99] O. Kupferman and M.Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5:245–263, 1999.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–230, 1984.

[MS95] D.E. Muller and P.E. Schupp. Simulating aternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton, and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[MW80] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2:90–121, 1980.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th Symposium on Principles of Programming Languages*, pages 179–190. ACM Press, 1989.

[QBSP96] S. Qadeer, R. K. Brayton, V. Singhal, and C. Pixley. Latch redundancy removal without global reset. In *Proc. Intl. Conference on Computer Design*, pages 432–439. IEEE Computer Society, 1996.

[Rab70] M.O. Rabin. Weakly definable relations and special automata. *Mathematical Logic and Foundations of Set theory*, 1970.

[Rab72] M.O. Rabin. *Automata on Infinite Objects and Church's Problem.* Number 13 in Regional Conference Series in Mathematics. American Mathematical Society, 1972.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.

[Ros92] R. Rosner. *Modular Synthesis of Reactive Systems.* PhD thesis, Weizmann Institute of Science, 1992.

[Saf88] S. Safra. On the complexity of omega-automata. In *Proc. 29th Symposium on Foundations of Computer Science*, pages 319–327. IEEE Computer Society, 1988.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic, *Journal of the ACM*, 32:733–749, 1985.

[SP94] V. Singhal and C. Pixley. The verification problem for safe replaceability. In *Proc. Conference on Computer-Aided Verification*, LNCS 818, pages 311–323. Springer-Verlag, 1994.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[Wol82] P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications.* PhD thesis, Stanford University, 1982.