Monitoring Off-the-Shelf Components*

A. Prasad Sistla, Min Zhou, and Lenore D. Zuck

University of Illinois at Chicago
{sistla, mzhou, lenore}@cs.uic.edu

Abstract. Software is being developed from off-the-shelf third party components. The interface specification of such a component may be under specified or may not fully match the user requirement. In this paper, we address the problem of customizing such components to particular users. We achieve this by constructing a monitor that monitors the component and detects any bad behaviors.

Construction of such monitors essentially involves synthesizing safety properties that imply a given property that is obtained from the interface specifications of the component and the goal specification of the user. We present various methods for synthesizing such safety properties when the given property is given by an automaton or a temporal logic formula. We show that our methods are sound and complete. These results are extensions of the results given in [11].

1 Introduction

The process of constructing software is undergoing rapid changes. Instead of a monolithic software development within an organization, increasingly, software is being assembled using third-party components (e.g., JavaBeans, .NET, etc.). The developers have little knowledge of, and even less control over, the internals of the components comprising the overall system.

One obstacle to composing agents is that current formal methods are mainly concerned with "closed" systems that are built from the ground up. Such systems are fully under the control of the user. Hence, problems arising from ill-specified components can be resolved by a close inspection of the systems. When composing agents use "off-the-shelf" ones, this is often no longer the case. Out of consideration for proprietary information, or in order to simplify presentation, companies may provide *incomplete specifications*. Worse, some agents may have no description at all except one that can be obtained by experimentation. Even if the component is completely specified, it may not fully satisfy the user requirements. Despite either of the cases, i.e., being ill-specified or mismatched-matched, "off-the-shelf" components might still be attractive enough so that the designer of a new service may wish to use them. In order to do so safely, the designer must be able to deal with the possibility that these components may exhibit undesired or unanticipated behavior, which could potentially compromise the correctness and security of the new system.

* This work is supported in part by the NSF grants CCR-9988884 and CCR-0205365.

E.A. Emerson and K.S. Namjoshi (Eds.): VMCAI 2006, LNCS 3855, pp. 222–236, 2006.

The main problem addressed in this paper is that of customizing ill-specified or slightly mismatched off-the-shelf components for a particular user. We assume that we are given the interface specification Φ_I of the off-the-shelf component and the goal specification Φ which denotes the user requirement. We want to design a module M which runs in parallel with the off-the-shelf component and monitors its executions. If the execution violates the user specification Φ then the monitor M indicates this so that corrective action may be taken. Our customization only consists of *monitoring* the executions. (Here we are assuming that violation of Φ is not catastrophic and can be remedied by some other means provided it is detected in time; for example, leakage of the credit card number in a business transaction can be handled by alerting the credit card company.) See [11] for a motivating example.

Our goal is to obtain a specification ϕ for the module M so that M composed with the off-the-shelf component implies Φ. Further more, we want ϕ to be a safety property since violations of such properties can be monitored. Once such a specification ϕ is obtained as an automaton it is straight forward to construct the monitor M. Essentially, M would run the automaton on the executions of the off-the-shelf component and detect violations. Thus, given Φ_I and Φ, our goal is to synthesize a safety property ϕ so that $\Phi_I \wedge \phi \rightarrow \Phi$, or equivalently $\phi \rightarrow (\neg\Phi_I \vee \Phi)$, is a valid formula.

We considered the above problem in our previous work [11]. In that work, we concentrated on obtaining ϕ as a deterministic Büchi automaton. There we showed that while there is always some safety property ϕ that guarantees $\phi \rightarrow (\neg\Phi_I \vee \Phi)$ (e.g., the trivially false property), in general, there is no "maximal" such safety property. We also synthesized a family of safety properties ϕ_k, such that the higher k is, the more "accurate" and costly to compute ϕ_k is. We also defined a class of, possibly infinite state, deterministic *bounded automata*, that accept the desired property ϕ. For these automata we proved a *restricted completeness* result showing that if $(\neg\Phi_I \vee \Phi)$ is specified by a deterministic Büchi automaton $\mathcal{A}$ and $L(\mathcal{A})$ is the language accepted by $\mathcal{A}$ then for every safety property S contained in $L(\mathcal{A})$ there exists a bounded automaton that accepts the S. (Actually, the paper [11] erroneously stated that this method is complete in general; however, this was corrected in a revised version [12] claiming only restricted completeness.)

In this paper we extend these earlier results as follows. We consider the cases where $\neg\Phi_I \vee \Phi$ is given as an automaton or as a LTL formula. For the former case, when $\neg\Phi_I \vee \Phi$ is described by an automaton, we describe two ways of synthesizing the safety properties as a Büchi automaton $\mathcal{B}$. The first method assumes that the given automaton $\mathcal{A}$ is a non-deterministic Büchi automaton and constructs a non-deterministic $\mathcal{B}$ from $\mathcal{A}$ by associating a counter with each state. The constructed automaton is much simpler than the one given in [11]. We also define a class of infinite state automata and show that all these automata accept safety properties contained in $L(\mathcal{A})$. We also prove a restricted completeness result for this case.

In the second method we assume that $\mathcal{A}$ is given as a deterministic Streett automaton. For this case, we give the construction of a class of possibly infinite state automata that accept safety properties contained in $L(\mathcal{A})$. This construction employs a pair of counters for each accepting pair in the accepting condition of $\mathcal{A}$. We show that this construction is sound and is also complete when $\mathcal{A}$ is deterministic. Since deterministic Streett automata are more powerful than deterministic Büchi automata, this method is provably more powerful than the one given in [11]. Also, we can obtain a complete method for synthesizing safety properties contained in the language of a given non-deterministic Büchi or Streett automaton by systematically converting it into an equivalent deterministic Streett automaton [14] and by employing the above method on the resulting automaton.

In the case that Φ_I, Φ, and hence $\neg\Phi_I \vee \Phi$, are given as LTL formulas, we give semantic and syntactic methods. The semantic method constructs the tableau, from which it constructs a non-deterministic Büchi automaton that accepts the desired safety property. The syntactic method converts the formula $\neg\Phi_I \vee \Phi$ into another formula that specifies a safety property which implies $\neg\Phi_I \vee \Phi$.

Outline. Section 2 contains definitions, notation, and outlines some prior results relevant to this work. Section 3 studies synthesis of safety from specifications given by non-deterministic Büchi automata and shows a partial completeness result. Section 4 studies synthesis of safety from specifications given by deterministic Streett automata and shows a completeness result. Section 5 studies synthesis of safety from specifications given by LTL formulae. Section 6 discusses related literature, and we conclude in Section 7.

2 Preliminaries

Sequences. Let S be a finite set. Let $\sigma = s_0, s_1, \ldots$ be a possibly infinite sequence over S. The length of σ, denoted as $|\sigma|$, is defined to be the number of elements in σ if σ is finite, and ω otherwise. If α_1 is a finite sequence and α_2 is a either a finite or a ω-sequence then $\alpha_1\alpha_2$ denotes the concatenation of the two sequences in that order.

For integers i and j such that $0 \leq i \leq j < |\sigma|$, $\sigma[i, j]$ denotes the (finite) sequence $s_i, \ldots s_j$. A *prefix* of σ is any $\sigma[0, j]$ for $j < |\sigma|$. We denote the set of σ's prefixes by $\mathit{Pref}(\sigma)$. Given an integer i, $0 \leq i < |\sigma|$, we denote by $\sigma^{(i)}$ the *suffix* of σ that starts with s_i.

For an infinite sequence $\sigma : s_0, \ldots$, we denote by $\inf(\sigma)$ the set of S-elements that occur in σ infinitely many times, i.e., $\inf(\sigma) = \{s : s_i = s \text{ for infinitely many } i\text{'s }\}$.

Languages. A *language* L over a finite alphabet Σ is a set of finite or infinite sequences over σ. When L consists only of infinite strings (sequences), we sometimes refer to it as an *ω-language*. For a language L, we denote the set of prefixes of L by $\mathit{Pref}(L)$, i.e.,

$$\mathit{Pref}(L) = \bigcup_{\sigma \in L} \mathit{Pref}(\sigma)$$

Following [6, 2], an ω-language L is a *safety property* if for every $\sigma \in \Sigma^\infty$:

$$Pref(\sigma) \subseteq Pref(L) \quad \Longrightarrow \quad \sigma \in L$$

i.e., L is a safety property if it is *limit closed* – for every ω-string σ, if every prefix of σ is a prefix of some L-string, then σ must be an L-string.

Büchi Automata. A *Büchi automaton* (NBA for short) $\mathcal{A}$ on infinite strings is described by a quintuple $(Q, \Sigma, \delta, q^0, F)$ where:

- Q is a finite set of states;
- Σ is a finite alphabet of symbols;
- $\delta\colon Q \times \Sigma \to 2^Q$ is a transition function;
- $q^0 \in Q$ is an initial state;
- $F \subseteq Q$ is a set of accepting states.

The *generalized transition function* $\delta^*\colon Q \times \Sigma^* \to 2^Q$ is defined in the usual way, i.e., for every state q, $\delta^*(q, \epsilon) = \{q\}$, and for any $\sigma \in \Sigma^*$ and $a \in \Sigma$, $\delta^*(q, \sigma a) = \cup_{q' \in \delta^*(q,\sigma)} \delta(q', a)$.

If for every $(q, a) \in Q \times \Sigma$, $|\delta(q, a)| = 1$, then $\mathcal{A}$ is called a *deterministic* Büchi automaton (or DBA for short).

Let $\sigma : a_1, \ldots$ be an infinite sequence over σ. A *run* r *of* $\mathcal{A}$ on σ is an infinite sequence $q^0, \ldots$ over Q such that:

- $q^0 = q_0$;
- for every $i > 0$, $q^i \in \delta(q^{i-1}, a_i)$;

A run r on a Büchi automaton is *accepting* if $\inf(r) \cap F \neq \emptyset$. The automaton $\mathcal{A}$ *accepts* the ω-string σ if it has an accepting run over σ (for the case of DBAs, the automaton has a single run over σ). The *language accepted by* $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of ω-strings that $\mathcal{A}$ accepts. A language L' is called ω*-regular* if it is an ω-language that is accepted by some (possibly non-deterministic) Büchi automaton.

A Büchi automaton $\mathcal{A}$ can also be used to define a *regular automaton* that is just like $\mathcal{A}$, only the acceptance condition of a run r is that its last state is accepting. We denote the regular language accepted by the regular version of $\mathcal{A}$ by $L_f(\mathcal{A})$.

Infinite-state. Büchi automata are defined just like Büchi automata, only that set of states may be infinite. We denote infinite-state DBAs by iDBAs, and infinite-state NBAs by iNBAs.

Streett Automata. A *Streett automaton* $\mathcal{S}$ on infinite strings is described by a quintuple $(Q, \Sigma, \delta, q^0, F)$ where Q, Σ, δ, and q^0 are just like in Büchi automata, and F is of the form $\cup_{i=1}^m (U_i, V_i)$ where each $U_i, V_i \subseteq Q$. A run r of $\mathcal{S}$ on $\sigma = a_1, \ldots$ is defined just like in the case of Büchi automaton. The run is *accepting* if, for every $i = 1, \ldots, m$, if $\inf(r) \cap U_i \neq \emptyset$ then $\inf(r) \cap V_i \neq \emptyset$, i.e., if some U_i states appears infinitely often in r, then some V_i states should also appear infinitely often in r.

Every Büchi automaton can be converted into a deterministic Streett automaton that recognizes the same ω-language ([14]).

Linear-time Temporal Logic. We consider LTL formulae over a set of atomic propositions Π using the boolean connectives and the temporal operators $\bigcirc$ (*next*), $\mathcal{U}$ (*until*), and $\mathcal{W}$ (*weak until* or *unless*). Let $\Sigma = 2^{\Pi}$. We define a satisfiability relation $\models$ between infinite sequences over Σ by:

For a proposition $p \in \Pi$, $\sigma \models p$ iff $p \in \sigma^0$
$\sigma \models \phi \vee \psi$ iff $\sigma \models \phi$ or $\sigma \models \psi$
$\sigma \models \neg\phi$ iff $\sigma \not\models \phi$
$\sigma \models \bigcirc \phi_1$ iff $\sigma^{(1)} \models \phi_1$
$\sigma \models \phi_1 \,\mathcal{U}\, \phi_2$ iff for some $i \geq 0$, $\sigma^{(i)} \models \phi_2$, and for all j, $0 \leq j < i$, $\sigma^{(j)} \models \phi_1$
$\sigma \models \phi_1 \,\mathcal{W}\, \phi_2$ iff for some $i \geq 0$, $\sigma^{(i)} \models \phi_1 \wedge \phi_2$, and for all j, $0 \leq j < i$, $\sigma^{(j)} \models \phi_1$

Note that the semantics of the unless operator is slightly different than the usual one, in requiring the ϕ_1 to hold on the state where ϕ_2 is first encountered.

For every LTL formula ϕ, we denote the set of atomic propositions that appear in ϕ by $Prop(\phi)$, and the ω-language that ϕ defines, i.e., the set of infinite sequences (models) that satisfy ϕ by $L(\phi)$.

Let ϕ be an LTL formula. We define the *closure* of ϕ, denoted by $cl(\phi)$ to be the minimal set of formulae that is closed under negation and includes ϕ, every subformula of ϕ, and for every subformula $\phi_1 \,\mathcal{W}\, \phi_2$ of ϕ the formula $\neg\phi_2 \,\mathcal{U}\, \neg\phi_1$. The *atoms* of ϕ, denoted by $at(\phi)$, is a subset of $2^{cl(\phi)-\emptyset}$ such that each atom $A \in at(\phi)$ is a maximally consistent subset of $cl(\phi)$ where for every $\psi = \phi_1 \,\mathcal{W}\, \phi_2 \in cl(\phi)$, $\psi \in A$ iff $(\neg\phi_2 \,\mathcal{U}\, \neg\phi_1) \notin A$. An *initial atom* is any atom that contains ϕ. The *tableau* of ϕ, $tab(\phi)$, is a graph $(at(\phi), R)$ whose nodes are $at(\phi)$, and a $(A_1, A_2) \in R$ iff the following all hold:

For every $\bigcirc \psi \in cl(\phi)$, $\bigcirc \psi \in A_1$ iff $\psi \in A_2$
For every $\psi_1 \,\mathcal{U}\, \psi_2 \in cl(\phi)$, $\psi_1 \,\mathcal{U}\, \psi_2 \in A_1$ iff $\psi_2 \in A_1$ or $\psi_1 \in A_1$ and $\psi_1 \,\mathcal{U}\, \psi_2 \in A_2$
For every $\psi_1 \,\mathcal{W}\, \psi_2 \in cl(\phi)$, $\psi_1 \,\mathcal{W}\, \psi_2 \in A_1$ iff $\psi_1, \psi_2 \in A_1$ or $\psi_1, \psi_1 \,\mathcal{W}\, \psi_2 \in A_2$

It is known (e.g., [9]) that ϕ is satisfiable iff $tab(\phi)$ contains a path leading from an initial atom into a maximally strongly connected component (MSCC) $\mathcal{C}$ such that for every $\psi_1 \,\mathcal{U}\, \psi_2 \in A \in \mathcal{C}$, there is an atom $B \in \mathcal{C}$ such that $\psi_2 \in B$.(Such MSCCs are called "fulfilling.") Similarly, it is also known (see, e.g., [18, 3]) how to construct a NBA (and, consequently, a deterministic Streett automaton) that recognizes $L(\phi)$.

3 Synthesizing Safety from Büchi Automata

In this section we study synthesis of safety properties from a given NBA. We fix a (possibly non-deterministic) Büchi automaton $\mathcal{A} = (Q_A, \Sigma, \delta_A, q_A^0, F_A)$. As shown in [11], unless $L(\mathcal{A})$ is already safety, there is no maximal safety property that is contained in $L(\mathcal{A})$ which is Büchi recognizable. We first show an infinite

chain of safety properties that are all Büchi recognizable, each contained in the next. We then present *bounded Büchi automaton*, an infinite-state version of NBAs, and show that each accepts a safety property in $L(\mathcal{A})$. We also show a partial completeness result, namely, that if $\mathcal{A}$ is deterministic then each safety property contained in $L(\mathcal{A})$ is accepted by some bounded automata.

Both constructions, of the chain of Büchi automata and the bounded automata, are much simplified versions of their counterparts described in [11].

3.1 Synthesis of Safety into Büchi Automata

We define a chain of safety properties $\{L_k(\mathcal{A})\}_{k>0}$ such that for every k, $L_k(\mathcal{A}) \subseteq L(\mathcal{A})$ and $L_k(\mathcal{A}) \subseteq L_{k+1}(\mathcal{A})$.

Let $k > 0$ be an integer. The ω-language $L_k(\mathcal{A})$ is a subset of $L(\mathcal{A})$, where every string has an accepting $\mathcal{A}$-run with the first accepting (F_A) state appearing within the first k states of the run, and any successive accepting states in the run are separated by at most k states. Formally, $w \in L_k(\mathcal{A})$ iff there exists an accepting $\mathcal{A}$ run $r_0, \ldots$ such that the set of indices $I = \{i \geq 0 : r_i \in F_A\}$ on w satisfying the following condition: for some $\ell < k$, $\ell \in I$, and for every $i \in I$, there is some $j \in I$ such that $i < j < i + k$.

For $k \geq 0$, let $\mathcal{B}_k \colon (Q_k, \Sigma, \delta_k, \langle q_A^0, k-1\rangle, Q_k)$ be a NBA where:

- $Q_k = Q_A \times \{0, 1, \cdots, k-1\}$
- $\langle q', i'\rangle \in \delta_k(\langle q, i\rangle, a)$ iff
  - $i > 0$, $q \notin F_A$, and $i' = i - 1$;
  - $i \geq 0$, $q \in F_A$, and $i' = k - 1$.

The automaton $\mathcal{B}_k$ simulates the possible runs of $\mathcal{A}$ on the input and uses a modulo k counter to maintain the number of steps within which an accepting state should be reached. Note that, when $q \notin F_A$, there are no outgoing transitions from the state $\langle q, 0\rangle$. Since all $\mathcal{B}_k$-states are accepting states, $L(\mathcal{B}_k)$ is a safety property. Also, from the construction it immediately follows that $L(\mathcal{B}_k) = L_k(\mathcal{A})$. We therefore conclude:

Lemma 1. *$L_k(\mathcal{A})$ is a safety property, and $L(\mathcal{B}_k) = L_k(\mathcal{A})$.*

3.2 Synthesis of Safety into Bounded Automata

The construction of the previous section is not complete in the sense that there are always safety properties in $L(\mathcal{A})$ that are not recognized by $\mathcal{A}_k$. We introduce bounded automata, a new type of Büchi automata, and show that (1) they only recognize safety properties in $L(\mathcal{A})$, and (2) when $\mathcal{A}$ is deterministic, they can recognize every safety property contained in $L(\mathcal{A})$.

Assume some (possible infinite) set Y. A *bounded automaton over $\mathcal{A}$ using Y* is a (i)NBA described by a tuple $\mathcal{N} \colon (Q_N, \Sigma, \delta_N, q_N^0, F_N)$ where:

- $Q_N \subseteq Y \times Q_A \times (\mathbb{N} \cup \{\infty\})$. Given a state $q_N = \langle r, q, i\rangle \in Q_N$, we refer to r as the Y component of q_N, to q as the $\mathcal{A} - state$ of q_N, and to i as the *counter* of q_N;

- For every $\langle r, q, i\rangle \in Q_N$, if $(q', r', i') \in \delta_N(\langle q, r, i\rangle, a)$ then the following all hold:
  - $q' \in \delta_A(q, a)$;
  - If $i = \infty$ then $i' = \infty$;
  - If $q \notin F_A$, then $i' < i$ or $i' = \infty$;
- q_N^0 is in $Y \times \{q_A^0\} \times \mathbb{N}$;
- $F_N = Y \times Q_A \times \mathbb{N}$.

Note that there are no outgoing from a state whose $\mathcal{A}$-state is not in F_A and whose counter is 0. Also, once a run reaches a state with counter value ∞, it remains in such states. Since the counters of states with non-accepting $\mathcal{A}$-states either decrease or become ∞, it follows that once a run reaches a rejecting state (i.e., a state in $Q_N \setminus F_N$), it remains there. It thus follows from [16] that $L(\mathcal{N})$ is a safety property. Consider an accepting $\mathcal{N}$ run. Since the counters of states can only decrease finitely many times from non-accepting $\mathcal{A}$-states, it follows that the run has infinitely many accepting $\mathcal{A}$-states. Thus, its projection on the $\mathcal{A}$-states is an accepting $\mathcal{A}$-run. We can therefore conclude:

Lemma 2. *For a bounded automaton $\mathcal{N}$ over $\mathcal{A}$, $L(\mathcal{N})$ is a safety property in $L(\mathcal{A})$.*

Lemma 2 shows that bounded automata over $\mathcal{A}$ accept only safety properties that are contained in $L(\mathcal{A})$. We next identify the safety properties in $L(\mathcal{A})$ that bounded automata accept.

Recall that for a Büchi automaton $\mathcal{A}$, $L_f(\mathcal{A})$ is the regular language defined by the regular version of $\mathcal{A}$. Let $S \subseteq L(\mathcal{A})$ be a safety property. Similarly to [11], we define:

- For a sequence $\sigma \in S$ and $\alpha \in \mathit{Pref}(\sigma)$, let $\mathit{min\_idx}(\sigma, \alpha) = \mathit{min}\{|\beta| : \alpha\,\beta \in L_f(\mathcal{A}) \cap \mathit{Pref}(\sigma)\}$;
- For any $\alpha \in \Sigma^*$, let $Z(\alpha, S) = \{\mathit{min\_idx}(\sigma, \alpha) : \sigma \in S \wedge \alpha \in \mathit{Pref}(\sigma)\}$

Note that if $\alpha \in L_f(\mathcal{A}) \cap \mathit{Pref}(\sigma)$, then $\mathit{min\_idx}(\alpha, a) = 0$ and $Z(\alpha, S) = \{0\}$. Similarly, if $\alpha \notin \mathit{Pref}(S)$, then $Z(\alpha, S) = \emptyset$. It is shown in [11] that for every $\alpha \in \Sigma^*$, $Z(\alpha, S)$ is finite.

For any $\alpha \in \Sigma^*$, we define

$$\mathit{idx}(\alpha, S) \;=\; \begin{cases} \max_{i \in Z(\alpha,S)}\{i\} & Z(\alpha, S) \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$$

Thus, $\mathit{idx}(\alpha, S) \in \mathbb{N}$ iff $\alpha \in \mathit{Pref}(S)$.

Let $S \subseteq L(\mathcal{A})$ be a safety property. A *bounded S-automaton over $\mathcal{A}$* is a bounded automaton over $\mathcal{A}$ using Σ^*, of the form $\mathcal{D}$: $(Q_D, \Sigma, \delta_D, q_D^0, F_D)$, where:

- $Q_D = \{\langle \alpha, q, i\rangle : \alpha \in \Sigma^*,\ q \in Q_A,\ i \in \{\mathit{idx}(\alpha, S)\, \infty\}\}$;
- For every $q \notin F_A$, if $\langle \alpha', q', i'\rangle \in \delta_D(\langle \alpha, q\, i\rangle, a)$ then the following all hold:
  1. $\alpha' = \alpha\, a$;
  2. $q' \in \delta_A(q, a)$;

3. $i' = \begin{cases} idx(\alpha\, a, S) & \text{if } i \neq \infty \text{ and } idx(\alpha\, a, S) < i \\ \infty & \text{otherwise} \end{cases}$

– $q_D^0 = \langle \epsilon, q_A^0, idx(\epsilon, S) \rangle$.
– $F_D = \Sigma^* \times Q_A \times \mathbb{N}$

Theorem 1 (Partial Completeness). *For a safety property $S \subseteq L(\mathcal{A})$, $L(\mathcal{D}) \subseteq S$, and if $\mathcal{A}$ is a deterministic automaton then $L(\mathcal{D}) = S$.*

Proof. For (1), We show that for any $\sigma \in \Sigma^\omega$, if $\sigma \notin S$, then $\sigma \notin L(\mathcal{D})$. Assume $\sigma \in \Sigma^\omega \setminus S$. Since S is a safety property, there exists an integer i such that every prefix $\alpha \prec \sigma$, of length i or more, $\alpha \notin \mathit{Pref}(S)$. Consider such a prefix α. Obviously, $idx(\alpha, S) = \infty$. Hence, in any run, after reading α, the counter of the state reached is ∞. It follows that $\sigma \notin L(\mathcal{D})$.

For (2), assume that $\sigma \in S$. Define a sequence $\{p_i\}_{i \geq 0}$ over Σ^* such that $p_0 = \epsilon$ and for every $i > 0$, $p_i = \sigma[0, i-1]$, i.e., $\{p_i\}_{i \geq 0}$ is the sequence of σ's prefices. Since $S \subseteq L(\mathcal{A})$, there exists an accepting $\mathcal{A}$-run $r_A^0, \cdots$ of $\mathcal{A}$ on σ. Consider now the sequence of $\mathcal{D}$ states $\alpha = \{\langle p_j, r_j^A, idx(p_j, S)\}_{j \geq 0}$. The first element in α is q_D^0. Consider now the case that $\mathcal{A}$ is deterministic. When the $\mathcal{A}$ state of an α-state is $\mathcal{A}$-accepting, its counter is 0; otherwise, the counter of the next α-state is lower. Thus, α is a $\mathcal{D}$-run. Finally, since $\sigma \in S$, every counter in α is non-∞, thus $\mathcal{D}$ is accepting. □

To see why the method is incomplete for general NBAs, consider the NBA $\mathcal{A}$ described in Figure 1. Obviously, $L(\mathcal{A}) = L_1 \cup L_2$ where $L_1 = \{a^i b \Sigma^\omega : i > 0\} \cup \{a^\omega\}$ and $L_2 = \{\Sigma^i c^\omega : i \geq 0\}$. Note that L_1 is generated by the sub-automaton consisting of $\{q_0, q_1, q_2, q_3\}$ and L_2 is generated by the sub-automaton consisting of $\{q_0, q_4, q_5\}$. The language L_1 is clearly a safety property, however, the above method cannot generate L_1, since any bounded automaton where the value of the counter in the initial state is k can only accept the L_1 strings of the form $\{a^i b \Sigma^\omega : 0 < i \leq k\} \cup \{a^\omega\}$.

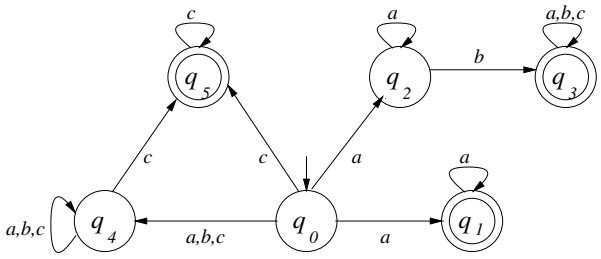

Fig. 1. Automaton $\mathcal{A}$

4 Synthesizing Safety from Streett Automata

In this section, we generalize the construction of the bounded automata of the previous section into *extended bounded automata* for synthesizing safety properties contained in the language of a given Streett automaton (SA). We show that

this construction is sound, and it is complete when the given automaton is a deterministic Streett Automaton (DSA). Since NBAs and SAs can be determinized into an equivalent deterministic Streett automata (DSAs) [14], the construction given in this section can be used to synthesize any safety property contained in the language accepted by a given NBA or SA, thus giving a complete system for these automata also.

Let $\mathcal{A} = (Q_A, \Sigma, \delta_A, q_A^0, F_A)$ be a SA. Assume that $F_A = \cup_{i=1}^{m}\{(U_i, V_i)\}$. Without loss of generality, assume that for every $i = 1, \ldots, m$, $U_i \cap V_i = \emptyset$. (If $U_i \cap V_i \neq \emptyset$, then U_i can be replaced by $U_i \setminus V_i$ without impacting the recognized language.)

Let Y be some (possibly infinite) set. An *extended bounded automaton* $\mathcal{E}$ *over* $\mathcal{A}$ *using* Y is a (i)NBA $(Q_E, \Sigma, \delta_E, q_E^0, F_E)$ where:

- $Q_E \subseteq Y \times Q_A \times (\mathbb{N} \cup \{\infty\})^{(2m)}$;
- For every $R = \langle r, q, i_1, j_1, \ldots, i_m, j_m\rangle \in Q_E$ and $a \in \Sigma$, if $R' \in \delta_E(R, a)$ where $R' = \langle r', q', i'_1, j'_1, \ldots, i'_m, j'_m\rangle$ the following all hold:
  - $q' \in \delta_A(q, a)$;
  - For every $k = 1, \ldots, m$:
    1. If $i_k = \infty$ then $i'_k = \infty$ and if $j_k = \infty$ then $j'_k = \infty$;
    2. if $q' \in U_k$ then, if $i_k > 0$ then $i'_k < i_k$, and if $i_k = 0$ then $i'_k = 0$ and $j'_k < j_k$.
    3. $q' \notin (U_k \cup V_k)$ then,if $i_k > 0$ then $i'_k \leq i_k$, and if $i_k = 0$ then $i'_k = 0$ and $j'_k \leq j_k$.
- $q_E^0 \in Y \times \{q_A^0\} \times \mathbb{N}^{(2m)}$;
- $F_E = Y \times Q_A \times \mathbb{N}^{(2m)}$.

Extended bounded automata are similar to bounded automata. However, states of extended bounded automata associate a pair of counters (i_k, j_k) for each accepting pair (U_k, V_k) in F_A, and the transition function is different. Just like in the case of bounded automata, when an extended automaton enters a rejecting state, it cannot re-enter an accepting state. It thus follows from [16] that extended bonded automata can only recognize safety properties.

Consider an accepting run $\rho = R_1, \ldots$ of $\mathcal{E}$, where for every $k \geq 0$, $R_k = \langle r_k, q_k, i_{k,1}, j_{k,1}, \ldots, i_{k,m}, j_{k,m}\rangle$. For $Q' \subseteq Q_A$ and $k \geq 0$, we say that a *Q'-state appears in R_k* if $q_k \in Q'$. Assume that for some $\ell = [1..m]$, U_ℓ appears infinitely many times in ρ's states, and let $k \geq 0$. Consider now the sequence of $i_{k',\ell}$ for $k' \geq k$. Since ρ is accepting, and there are infinitely many $R_{k'}$ where a U_ℓ-state appears, $i_{k',\ell}$ never increases until some V_ℓ-state appears, and decreases with each appearance of of a U_ℓ-state. Once $i_{k',\ell}$ becomes zero, the value of $j_{k',\ell}$ decrease with each appearance of a U_ℓ-state. Thus, a V_ℓ state must appear in ρ after R_k. It thus follows that R, projected onto its Q_A-states, is an accepting run of $\mathcal{A}$. We can therefore conclude:

Theorem 2 (Soundness). *For every extended bounded automaton $\mathcal{E}$ over $\mathcal{A}$, the language recognized by $\mathcal{E}$ is a safety property that is contained in $L(\mathcal{A})$.*

We now turn to prove completeness, i.e., we show that if $\mathcal{A}$ is deterministic then *every* safety property in $L(\mathcal{A})$ is recognized by some extended bounded

automaton. We fix some safety property $S \in L(\mathcal{A})$ and show it is recognized by some extended bounded automaton.

For the proof we define and prove some properties of (finitely branching) infinite labeled trees. For details on the definition and proofs see the complete version of this paper in [15]. For space reasons, we only outline the main ideas here.

A tree is *finitely branching* if each node has finitely many children. Consider a finitely branching tree T with a labeling function ℓ from T's nodes that labels each nodes with one of three (mutually disjoint) labels, $\mathtt{lab}_1$, $\mathtt{lab}_2$, and $\mathtt{lab}_3$. An infinite path in T is *acceptable* with respect to ℓ over $(\mathtt{lab}_1, \mathtt{lab}_2, \mathtt{lab}_3)$ if it either contains only finitely many nodes with $\mathtt{lab}_1$-labels ($\mathtt{lab}_1$-nodes), or it contains infinitely many nodes with $\mathtt{lab}_2$ labels ($\mathtt{lab}_2$-nodes). The tree T is acceptable with respect to ℓ if each of its infinite paths is, and it is acceptable if it is acceptable with respect to some labeling function as above.

A *ranking function* on T is a function associating each node with a non-negative integer. Pair (ρ_1, ρ_2) of ranking functions is *good* if for every two nodes n and n' in T such that n is the parent of n', the following hold:

1. If n' is a $\mathtt{lab}_1$-node and $\rho_1(n) > 0$ then $\rho_1(n) > \rho_1(n')$.
2. If n' is a $\mathtt{lab}_1$-node and $\rho_1(n) = 0$ then $\rho_1(n') = 0$ and $\rho_2(n) > \rho_2(n')$.
3. If n' is a $\mathtt{lab}_3$-node and $\rho_1(n) > 0$ then $\rho_1(n) \geq \rho_1(n')$.
4. If n' is a $\mathtt{lab}_3$-node and $\rho_1(n) = 0$ then $\rho_1(n') = 0$ and $\rho_2(n) \geq \rho_2(n')$.

Theorem 3. **[15]** *A labeled finitely-branching tree T is acceptable iff there is a good pair of ranking functions for it.*

We can now prove the completeness theorem:

Theorem 4. *Let $\mathcal{A}$ be a deterministic Streett automaton and $S \subseteq L(\mathcal{A})$ be a safety property. There exists an extended bounded automaton $\mathcal{B}$ such that $L(\mathcal{B}) = S$.*

Proof. Consider the finitely branching tree T whose set of nodes is $\{(\alpha, q) : \alpha \in \mathit{Pref}(S),\ q = \delta_A^*(q_A^0, \alpha)\}$, its root is (ϵ, q_A^0), and for any two nodes $n = (\alpha, q)$ and $n' = (\alpha', q')$, n' is a child of n iff for some $a \in \Sigma$, $\alpha' = \alpha\, a$ and $q' \in \delta_A(q, a)$. Then, for an infinite path π starting from the root, its projection on its second component is an accepting run of $\mathcal{A}$ on the string which is the limit of its first projection (on $\mathit{Pref}(S)$).

For every $k = 1, \ldots, m$, let ℓ_k be a labeling of T by the labels $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{n}$ such that every U_k node in T (i.e., a node whose second component is in U_k) is labeled with $\mathbf{u}$, every V_k node is labeled with $\mathbf{v}$, and every node that is in neither U_k or V_k is labeled with $\mathbf{n}$. It follows that T is acceptable with respect to the labeling ℓ_k over $(\mathbf{u}, \mathbf{v}, \mathbf{n})$.

It follows from Theorem 3 that for every $k = 1, \ldots, m$, there exists a pair $(\rho_{k,1}, \rho_{k,2})$ of ranking functions such that for every two nodes $n = (\alpha, q)$ and $n' = (\alpha', q')$ such that n is the parent of n', the following all hold:

- If n' is a U_k node then
  - If $\rho_{k,1}(n) > 0$ then $\rho_{k,1}(n') < \rho_{k,1}(n)$;
  - If $\rho_{k,1}(n) = 0$ then $\rho_{k,1}(n') = 0$ and $\rho_{k,2}(n') < \rho_{k,2}(n)$;

- If n' is neither a U_k nor a V_k node, then
  - If $\rho_{k,1}(n) > 0$ then $\rho_{k,1}(n') \leq \rho_{k,1}(n)$;
  - If $\rho_{k,1}(n) = 0$ then $\rho_{k,1}(n') = 0$ and $\rho_{k,2}(n') \leq \rho_{k,2}(n)$;

We now define an extended bounded automaton $\mathcal{E} = (Q_E, \Sigma, \delta_E, q_E^0, F_E)$ where:

- For every $(\alpha, q, i_1, j_1, ..., i_m, j_m) \in Q_E$,
  1. $q = \delta_A^*(q_A^0, \alpha)$;
  2. If $(\alpha, q) \in T$ then, for each $k = 1, \ldots, m$, $i_k = \rho_{k,1}((\alpha, q))$ and $j_k = \rho_{k,2}((\alpha, q))$;
  3. If $(\alpha, q) \notin T$ then, for each $k = 1, \ldots, m$, $i_k = j_k = \infty$. Note that this definition guarantees that for every finite string α and $\mathcal{A}$-state q, there is a unique $\mathcal{E}$-state whose first two coordinates are α and q.
- For every state $R = (\alpha, q, \ldots) \in Q_E$ and symbol $a \in \Sigma$, $\delta_E(R, a)$ is the unique Q_E state whose first two coordinates are $\alpha\ a$ and $\delta_A(q, a)$;
- q_E^0, the initial state of $\mathcal{E}$, is the unique state whose first two coordinates are ϵ and q_A^0;
- F_E is the set of states such that for each $k = 1, \ldots, m$, $i_k, j_k \neq \infty$.

From the properties of the ranking functions it now follows that $L(\mathcal{E}) = S$. □

5 Synthesizing Safety from Temporal Formulae

In Sections 3 and 4 we describe how to synthesize safety properties from NBAs and SAs. In this section we discuss how to directly synthesize a safety property from an LTL formula.

Let ϕ be a LTL formula. As discussed in Section 2, one can construct $tab(\phi)$ and then an NBA, or a DSA, that recognizes $L(\phi)$, from which any of the approaches described in Section 3 can be used. However, such approaches may drastically alter $tab(\phi)$. In this section we describe two methods to obtain safety properties from $tab(\phi)$ while preserving its structure. The first is semantics based, and consists of augmentation to $tab(\phi)$. The second is syntactic based, and uses $tab(\phi)$ for monitoring purposes.

5.1 A Semantic Approach

Let ϕ be an LTL formula and consider $tab(\phi) = (at(\phi), R)$. We construct an expanded version of the tableau. We outline the construction here and defer formal description to the full version of the paper:

With each formula $\psi = \psi_1\ \mathcal{U}\ \psi_2 \in cl(\phi)$ we associate a counter c_ψ that can be "active" or "inactive". (Thus, we split atoms into new states, each containing the atoms and the counters.) If $(A_1, A_2) \in R$, then in the new structure, if the counter associated with ψ in A_1 is non-zero and active, and $\psi_2 \notin A_2$, then the counter is decremented; if $\psi_2 \in A_2$, the counter becomes inactive. If the value of the counter is zero, the transition is disabled. If the counter is inactive and $\psi \in A_2$, then the counter becomes active and is replenished to some constant k.

An initial node of the expanded tableau is one which corresponds to an initial atom, where only counters of $\mathcal{U}$ formulae that are in the node are active (and set to k) and the others inactive. Obviously, a "good" path in the new structure is one that starts at an initial node, and for every $\mathcal{U}$ formula in $cl(\phi)$, either the counter of the formula is infinitely many often inactive, or it is infinitely often equal to k.

In the full version of the paper we will show how the new structure defines a NBA that accepts safety properties in $L(\phi)$.

5.2 A Syntactic Approach

Let ϕ be an LTL formula where all negations are at the level of propositions. This is achieved by the following rewriting rules:

$$\begin{array}{lll} \neg(\psi_1 \vee \psi_2) \Longrightarrow (\neg\psi_1 \wedge \neg\psi_2) & & \neg(\psi_1 \wedge \psi_2) \Longrightarrow (\neg\psi_1 \vee \neg\psi_2) \\ \neg \bigcirc \psi \Longrightarrow \bigcirc\neg\psi & & \neg(\psi_1 \,\mathcal{U}\, \psi_2) \Longrightarrow (\neg\psi_2 \,\mathcal{W}\, \neg\psi_1) \\ \multicolumn{3}{c}{\neg(\psi_1 \,\mathcal{W}\, \psi_2) \Longrightarrow (\neg\psi_2 \,\mathcal{U}\, \neg\psi_1)} \end{array}$$

Let k be a positive integer. We construct an LTL formula ϕ_k by replacing each sub-formula of the form $\psi_1 \,\mathcal{U}\, \psi_2$ appearing ϕ with $\psi_1 \,\mathcal{U}_{\leq k}\, \psi_1$ where $\mathcal{U}_{\leq k}$ is the *bounded until* operator (i.e., $\mathcal{U}_{\leq k}$ guarantees its right-hand-side within k steps.) The following theorem, which gives a syntactic method for synthesizing safety properties, can be proven by induction on the length of ϕ. The monitor for ϕ_k can thenb be built by obtaining its tableau.

Theorem 5. *Let ϕ be a temporal formula, let k be positive integer, and let ϕ_k be as defined above. Then $L(\phi_k)$ is a safety property which implies ϕ.*

6 Related Work

As indicated in the introduction, in [11, 12] we studied the problem of synthesizing safety from Büchi specifications and presented a solution that satisfies restricted completeness in the sense that not all safety property can be synthesized. The work here presents a solution that is both simpler and complete, namely, given a NBA $\mathcal{A}$, the construction here generates any safety property that is in $L(\mathcal{A})$. In addition, the work here presents synthesis of safety property directly from LTL properties. These methods are much simpler than the ones given in [11].

Similar in motivation to ours, but much different in the approach, is the work in [13]. There, the interaction between the module and the interface is viewed as a 2-player game, where the interface has a winning strategy if it can guarantee that no matter what the module does, Φ is met while maintaining Φ_I. The work there only considers determinisic Büchi automata. The approach here is to synthesize the interface behavior, expressed by (possibly) non-determinisic automata, before constructing the module.

Some of the techniques we employ are somewhat reminiscent of techniques used for verifying that a safety property described by a state machine satisfies a

correctness specification given by an automaton or temporal logic. For example, simulation relations/state-functions together with well-founded mappings [5, 1, 17] have been proposed in the literature for this purpose. Our bounded automata use a form of well-founded mappings in the form of positive integer values that are components of each state. (This is as it should be, since we need to use some counters to ensure that an accepting state eventually appears.) However, here we are not trying to establish the correctness of a given safety property defined by a state machine, but rather, we are deriving safety properties that are contained in the language of an automaton.

In [7, 8] Larsen et al propose a method for turning an implicit specification of a component into an explicit one, i.e., given a context specification (there, a process algebraic expression with a hole, where the desired components needs to be plugged in) and an overall specification, they fully automatically derive a temporal safety property characterizing the set of all implementations which, together with the given context, satisfy the overall specification. While this technique has been developed for component synthesis, it can also be used for synthesizing optimal monitors in a setting where the interface specification Φ_I and the goal specification Φ are both safety properties. In this paper, we do not make any assumptions on Φ_I and Φ. They can be arbitrary properties specified in temporal logic or by automata. We are aiming at exploiting liveness guarantees of external components (contexts), in order to establish liveness properties of the overall system under certain additional safety assumptions, which we can run time check (monitor). This allows us to guarantee that the overall system is as live as the context, as long as the constructed monitor does not cause an alarm.

There has been much on monitoring violations of safety properties in distributed systems. In these works, the safety property is typically explicitly specified by the user. Our work is more on deriving safety properties from component specifications than developing algorithms for monitoring given safety properties. In this sense, the approach to use safety properties for monitoring that have been automatically derived by observation using techniques adapted from automata learning (see [4]) is closer in spirit to the technique here. Much attention has since been spent in optimizing the automatic learning of the monitors [10]. However, the learned monitors play a different role: whereas the learned monitors are good, but by no means complete, sensors for detecting unexpected anomalies, the monitors derived with the techniques of this paper *imply* the specifying property as long as the guarantees of the component provider are true.

7 Conclusions and Discussion

In this paper, we considered the problem of customizing a given, off-the-shelf, reactive component to user requirements. In this process, we assume that the reactive module's external behavior is specified by a formula Φ_I and the desired goal specifications is given by a formula Φ. We presented methods for obtaining a safety specification ϕ so that $\phi \rightarrow (\neg\Phi_I \vee \Phi)$ by synthesizing (possibly infinite-state) NBAs for monitoring off-the-shelf components.

More specifically, we considered three different cases. The first one is when $(\neg\Phi_I \vee \Phi)$ is given as a non-deterministic Büchi automaton. For this case, the synthesized safety property is also given as a non-deterministic automaton. This method is shown to to be sound and complete when the given automaton is deterministic. This method is simpler than the one given in [11]. The second case is when $(\neg\Phi_I \vee \Phi)$ is given as a Streett automaton. In this case also, we gave a sound method for synthesizing safety properties contained in the language of the automaton, The method is also shown to be complete if the given automaton is a deterministic Streett Automaton. Since every Büchi automaton and Streett automaton can be converted algorithmically into an equivalent deterministic Streett automaton, this method gives us a complete system for synthesizing any safety property contained in the language of a given Büchi automaton or Streett automaton. The last case is when $\neg\Phi_I \vee \Phi$ is a LTL formula. In this case, we outlined a semantic method that works directly with tableux associated with formulae, without converting the tableux into automata. We also gave a syntactic method for this.

For our automata to be useful, they should be recursive, i.e., their set of states and their transition functions should be recursive functions. For monitoring purposes we need not explicitly compute the automaton and keep it in memmory, rather, we only need to maintain its current state and, whenever a new input arrives, we can use the transition function to compute the next state. For a non-dterministic automaton, we need to maintain the set of reachable states. Because of finite non-determinism, this set will be finite and is also computed on-the-fly after each input.

It is to be noted that the counters that are used only count down after occurrence of some input symbols. If the off-the-shelf component never responds, the autoamton remains in its current (good) state and permits such computations. This problem can be overcome by assuming that clock ticks are inputs to the automaton as well, and allowing a counter to count down with (some or all) clockticks. There are also other possible solutions to this problem.

We implemented a preliminary version of the method given in Section 3. It will be interesting to implement the general version given in Section 4 and apply it to practical problems. It will also be interesting to investigate how the counter values, given in these constructions, can be computed as functions of the history seen thus far. Real-time implementation of such systems need to be further investigated.

References

1. M. Abadi and L. Lamport. The existence of state mappings. In *Proceedings of the ACM Symposium on Logic in Computer Science*, 1988.
2. B. Alpern and F. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
3. E. A. Emerson and A. P. Sistla. Triple exponential decision procedure for the logic CTL*. In *Workshop on the Logics of Program, Carnegie-Mellon University*, 1983.

4. H. Hungar and B. Steffen. Behavior-based model construction. *STTT*, 6(1):4–14, 2004.
5. B. Jonsson. Compositional verification of distributed systems. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, 1987.
6. L. Lamport. Logical foundation, distributed systems- methods and tools for specification. *Springer-Verlag Lecture Notes in Computer Science*, 190, 1985.
7. K. Larsen. Ideal specification formalisms = expressivity + compositionality + decidability + testability + ... In *Invited Lecture at CONCUR 1990, LNCS 458*, 1990.
8. K. Larsen. The expressive power of implicit specifications. In *ICALP 1991, LNCS 510*, 1991.
9. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. Princ. of Prog. Lang.*, pages 97–107, 1985.
10. T. Margaria, H. Raffelt, and B. Steffen. Knowledge-based relevance filtering for efficient system-level test-based model generation (to appear). *Innovations in Systems and Software Engineering, a NASA Journal, Springer Verlag.*
11. T. Margaria, A. Sistla, B. Steffen, and L. D. Zuck. Taming interface specifications. In *CONCUR2005*, pages 548–561, 2005.
12. T. Margaria, A. Sistla, B. Steffen, and L. D. Zuck. Taming interface specifications. In *www.cs.uic.edu/~sistla*, 2005.
13. A. Pnueli, A. Zaks, and L. D. Zuck. Monitoring interfaces for faults. In *Proceedings of the 5^{th} Workshop on Runtime Verification (RV'05)*, 2005. To appear in a special issue of ENTCS.
14. S. Safra. On the complexity of ω-automata. In *29th annual Symposium on Foundations of Computer Science, October 24–26, 1988, White Plains, New York*, pages 319–327. IEEE Computer Society Press, 1988.
15. A. Sistla, M. Zhou, and L. D. Zuck. Monitoring off-the-shelf components. A companion to the VMCAI06 paper, `www.cs.uic.edu/~lenore/pubs`, 2005.
16. A. P. Sistla. On characterization of safety and liveness properties in temporal logic. In *Proceedings of the ACM Symposium on Principle of Distributed Computing*, 1985.
17. A. P. Sistla. Proving correctness with respect to nondeterministic safety specifications. *Information Processing Letters*, 39:45–49, 1991.
18. M. Vardi, P. Wolper, and A. P. Sistla. Reasoning about infinite computations. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 1983.