---

**Proceedings Paper:**
Dimitrova, R., Finkbeiner, B. and Rabe, M.N. (2012) Monitoring temporal information flow. In: Margaria, T. and Steffen, B., (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change. International Symposium On Leveraging Applications of Formal Methods, Verification and Validation - ISoLA 2012, 15-18 Oct 2012, Heraklion, Crete, Greece. Lecture Notes in Computer Science, 1 (7609). Springer , pp. 342-357. ISBN 9783642340253

https://doi.org/10.1007/978-3-642-34026-0_26

---

# Monitoring Temporal Information Flow

Rayna Dimitrova, Bernd Finkbeiner, and Markus N. Rabe

Universität des Saarlandes, Germany

**Abstract.** We present a framework for monitoring information flow in security-critical reactive systems, such as communication protocols, cell phone apps, document servers and web browsers. The secrecy requirements in such systems typically vary over time in response to the interaction with the environment. Standard notions of secrecy, like noninterference, must therefore be extended by specifying precisely *when* and *under what conditions* a particular event needs to remain secret. Our framework is based on the temporal logic SecLTL, which combines the standard temporal operators of linear-time temporal logic with the modal *Hide* operator for the specification of information flow properties. We present a first monitoring algorithm for SecLTL specifications, based on a translation of SecLTL formulas to alternating automata, and identify open research questions and directions for future work.

## 1 Introduction

Runtime monitoring and enforcement of security properties has been an active area of research in the last four decades [1], and its importance continues to grow as *security-critical systems* such as communication protocols, cell phone apps, document servers and web browsers become more and more ubiquitous. The canonical property of interest in such systems is *noninterference* [2], which requires that the public output of the system does not depend on secret input. In this paper we address the problem of monitoring information flow in *reactive systems*. We argue that in the realm of reactive systems, considering classical noninterference is of limited interest: the secrecy requirements of a reactive system typically vary over time in response to the interaction with the environment.

Consider, for example, the flow of information in a system for managing clinical data. Ethical guidelines, such as those by the Caldicott Committee [3], state that patient information is confidential and should not be disclosed without the patient's consent unless justified for a lawful purpose. For example, a release of information without the patient's consent may be allowed (and even required) in certain cases of food poisoning and other notifiable diseases. On the other hand, many secondary uses of the information, for example for research purposes, are only allowed while the patient has given and not (yet) revoked explicit consent.

Due to the nature of information flow, monitoring mechanisms for noninterference and related properties must not only consider the monitored trace, but also additional traces that were *not* observed. For example, if the result of a medical test is to be kept secret, and the test result turns out to be positive in

the monitored trace, then we must not only track the computation path corresponding to the positive result, but also the computation path corresponding to the negative result in order to check that both traces are observably equivalent. To reduce the runtime overhead resulting from having to analyze parts of the actual system, hybrid approaches that combine dynamic and static analysis techniques have been developed [4, 5]. There exist, however, no means of imposing restrictions on which parts the system should be considered while still performing a semantically justified security analysis. This can be achieved by specifying precisely *when* and *under what conditions* noninterference has to hold.

In runtime verification, the usual specification language to describe such temporal contexts is temporal logic. In this paper, we therefore propose an approach for runtime monitoring of information flow in reactive systems that integrates the dynamic analysis of information flow into the monitoring of temporal properties. Our approach is based on the temporal logic SecLTL, which we recently introduced as a specification language for model checking [6]. SecLTL extends linear-time temporal logic (LTL) with the *Hide* operator $\mathcal{H}$ for the specification of information flow properties. The SecLTL formula $\mathcal{H}_{H,I,O}\,\varphi$ specifies that a certain secret, expressed as the current valuation of the variables in $H$, will not become observable before the endcondition $\varphi$ evaluates to true. The observer from whom we wish to hide the secret is characterized by the subsets $I$ and $O$ of the input and output variables that are visible to the observer. Applications of the Hide operator can be embedded into a temporal context, for example in the formula $\square\,(\neg c \rightarrow \mathcal{H}_{\{t\},I,O}\,\mathit{false})$, which specifies that *whenever* the patent does not give consent $c$ to release the information, then the test result $t$ must be kept confidential *forever* with respect to the variables in $I$ and $O$, representing the interface to potential secondary users of the information.

The challenge in using SecLTL as a specification language for runtime monitors is to integrate the monitoring of noninterference into the runtime verification of the temporal formula. In the paper, we show that a seamless integration is indeed possible by using alternating automata as an intermediate data structure. Alternating automata combine the disjunctive branching of nondeterministic automata with the conjunctive branching of universal automata. As a result, alternating automata are exponentially more succinct than nondeterministic or universal automata. LTL specifications can be translated in linear time into equivalent alternating automata that closely match the structure of the formula: the states of the automaton correspond to subformulas of the specification [7]. In monitoring, this conciseness can be exploited by an efficient on-the-fly construction, which delays the unfolding of the alternating automaton until new positions of the trace become available [8].

The universal branching available in alternating automata can also be used to concisely express the noninterference requirements in SecLTL specifications. Overall, the automaton for a SecLTL formula has the same structure as the automaton for an LTL formula. If the Hide operator occurs as a subformula, we need to check that *all* alternative traces (corresponding to different values of the secrets) in the system result in the same observation. To verify this condition, the

automaton branches *universally* into a separate check for *each* alternative trace, where the subautomaton for each alternative trace keeps track of both the state of the system for the main trace and the state of the system for the alternative trace, and ensures that the observations are the same until the endcondition becomes true on the main trace.

The embedding of the noninterference check into the alternating automaton leads to an efficient monitoring algorithm. Using the standard on-the-fly unfolding technique [8], we only consider that part of the automaton that corresponds to the monitored trace. In particular, the decision which alternative traces to track is delayed until the moment when the temporal context has already been evaluated with respect to the currently available prefix of the monitored trace.

In the following sections, we present the ingredients of our monitoring approach in more detail. In Section 2, we formalize our system model and review the syntax and semantics of SecLTL. In Section 3 we describe the translation of SecLTL specifications to alternating automata. Based on this foundation, we then present our monitoring algorithm for SecLTL specifications. We conclude with a discussion of open research questions and future directions in Section 4.

## 2 The Specification Language SecLTL

### 2.1 System Model

**Definition 1.** *A* transition system $\mathcal{S} = (S, s^0, \mathcal{V}_\mathcal{I}, \mathcal{V}_\mathcal{O}, \Sigma, \delta)$ *consists of:*

- *a finite set of states $S$ with an initial state $s^0$,*
- *finite sets $\mathcal{V}_\mathcal{I}$ and $\mathcal{V}_\mathcal{O}$ of boolean input and output variables respectively, with $\mathcal{V}_\mathcal{I} \cap \mathcal{V}_\mathcal{O} = \emptyset$, and alphabet $\Sigma$ defined as $\Sigma = 2^{(\mathcal{V}_\mathcal{I} \cup \mathcal{V}_\mathcal{O})}$,*
- *a transition function $\delta : S \times \Sigma \to S$, which is a partial function.*

*We consider* input-enabled systems*, that is, we require for every $s \in S$ and $a \in 2^{\mathcal{V}_\mathcal{I}}$ that there exists an $o \in 2^{\mathcal{V}_\mathcal{O}}$ such that $\delta(s, a \cup o)$ is defined. We define the size of the transition system $\mathcal{S}$ as $|\mathcal{S}| = |S| + |\Sigma|$.*

For a set $A$, $A^*$ is the set of all finite sequences of elements of $A$ and $A^\omega$ is the set of all infinite sequences of elements of $A$. For a finite or infinite sequence $\pi$ of elements of $A$ and $i \in \mathbb{N}$, $\pi[i]$ is the $(i+1)$-th element of $\pi$, $\pi[0, i)$ is the prefix of $\pi$ of up to (excluding) position $i$, $\pi[0, i]$ is the prefix of $\pi$ up to (including) position $i$ and, if $\pi$ is infinite, $\pi[i, \infty)$ is its infinite suffix starting at position $i$. We denote the length of a sequence $\pi$ with $|\pi|$ (where $|\pi| = \infty$ for $\pi \in A^\omega$).

**Definition 2 (Trace).** *A* trace *in a transition system $\mathcal{S} = (S, s^0, \mathcal{V}_\mathcal{I}, \mathcal{V}_\mathcal{O}, \Sigma, \delta)$ is a finite or infinite sequence $\pi$ of elements of $\Sigma$: $\pi \in \Sigma^* \cup \Sigma^\omega$.*

**Definition 3 (Execution).** *Given a state $s \in S$ and a finite or infinite trace $\pi$, there exists at most one (finite or infinite, respectively) sequence of states $s_0, s_1, \ldots$ such that $s_0 = s$ and $s_i = \delta(s_{i-1}, \pi[i-1])$ for all $0 < i < |\pi|$.*

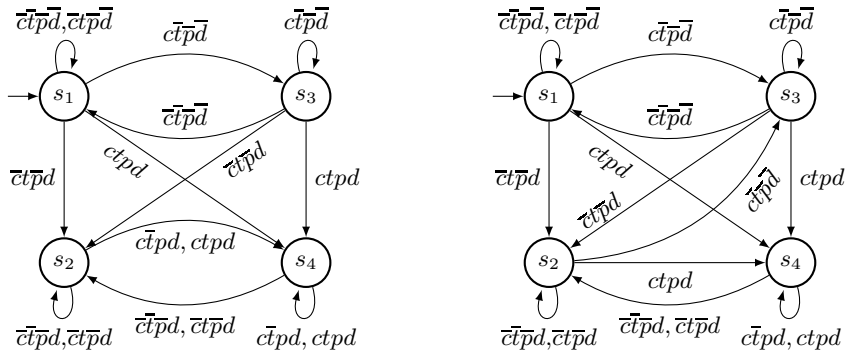
*We call this sequence of states (whenever it exists) an execution of $\mathcal{S}$ from $s$ on $\pi$ and denote it with $\mathsf{Exec}_\mathcal{S}(s, \pi)$. Given a state $s$, we denote the set of*

*infinite (finite) traces in $\mathcal{S}$ for which an execution of $\mathcal{S}$ from $s$ exists (i.e., for which $\mathsf{Exec}_{\mathcal{S}}$ is defined for the state $s$) by $\mathsf{Traces}_{\mathcal{S},s}$ (respectively $\mathsf{TracesFin}_{\mathcal{S}.s}$).*

*Example 1.* We use a simple application for managing clinical data as our running example. The application processes medical test results for a patient and reports the results in accumulated form, here simply stating whether or not some test has been positive, to an external agency. The application takes into account whether the patient agrees to the release of information. Figures 1(a) and 1(b) show two transition systems modeling variations of such an application. In both $\mathcal{S}_A$ and $\mathcal{S}_B$, the set of input variables is $\mathcal{V}_\mathcal{I} = \{c, t\}$ and the set of output variables is $\mathcal{V}_\mathcal{O} = \{p, d\}$. Thus, the alphabet of edge labels is $\Sigma = 2^{\{t,c,p,d\}}$.

The variable $t$ indicates the positive or negative outcome of the current test, the variable $p$ indicates whether the system reports that some test was positive, the variable $c$ indicates the patient's current consent status, and the variable $d$ indicates whether the patient is included in an ongoing drug study. For clarity, we use the valuations of the boolean variables in $\mathcal{V}_\mathcal{I} \cup \mathcal{V}_\mathcal{O}$ to represent the elements of $\Sigma$, for example the edge label $c\bar{t}pd$ stands for $\{c, p, d\}$.

If the patient consents to the release of information, $p$ is truthfully set to *true* if some test result has been positive. If there is no consent, $p$ is always set to *false*. The behaviors of the two systems differ with respect to test results that occurred while the patient did not consent to the release of information. In $\mathcal{S}_A$, once an execution reaches state $s_2$ or state $s_4$ (when some test result is positive), it stays in the set of states $\{s_2, s_4\}$ forever. The test results are therefore recorded accurately, even if there is no consent to release the information. System $\mathcal{S}_B$, on the other hand, goes to state $s_3$ whenever the patient changes the consent status




(a) Transition system $\mathcal{S}_A$ (b) Transition system $\mathcal{S}_B$

**Fig. 1.** Transition systems with input variables $c, t$ and output variables $p, d$. They model two simple systems for managing clinical data, which report on the accumulated result of a series of medical tests administered to a patient. The input variable $t$ indicates the positive or negative outcome of the current test, the output variable $p$ indicates whether the system reports that some test in the past has been positive, the output variable $d$ indicates whether the patient is included in a drug study, the input variable $c$ indicates whether the patient currently consents to the release of information.

from *false* to *true* (i.e., $c$ becomes *true*) and the current test result is negative, thus discarding the information about any positive test results up to that point. In addition to that, in this case, the patient's participation in the drug study is suspended until possibly a positive test result comes in the future.

The patient can be included in the drug study (i.e, the variable $d$ can be set to *true*) only if some test result was positive. As long as the patient does not consent to the release of information he may or may not be included in the drug study. In the latter case, positive test results are not recorded.

A possible finite trace of $\mathcal{S}_A$ is $\pi = c\bar{t}\overline{p}\overline{d}, \overline{c}\bar{t}\overline{p}\overline{d}, \overline{c}t\overline{p}d, c\bar{t}pd$. The corresponding execution of $\mathcal{S}_A$ from the initial state $s_1$ is $\mathsf{Exec}_{\mathcal{S}_A}(s_1, \pi) = s_1, s_3, s_1, s_2, s_4$. This trace is not possible in the system $\mathcal{S}_B$ (i.e., there exists no corresponding execution), because there is no transition for $c\bar{t}pd$ from state $s_2$ in $\mathcal{S}_B$.

### 2.2 SecLTL: Syntax and Semantics

The logic SecLTL, introduced in [6], extends LTL with the *Hide* operator $\mathcal{H}$.

Formally, the SecLTL formulas over a set of input variables $\mathcal{V}_\mathcal{I}$ and a set of output variables $\mathcal{V}_\mathcal{O}$ are defined according to the grammar below. Here, $p \in \mathcal{V}_\mathcal{I} \cup \mathcal{V}_\mathcal{O}$, $\varphi$ and $\psi$ are SecLTL formulas, $H \subseteq \mathcal{V}_\mathcal{I}$ and $I \subseteq \mathcal{V}_\mathcal{I}$ are sets of input variables with $H \subseteq I$, and $O \subseteq \mathcal{V}_\mathcal{O}$ is a set of output variables:

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \psi \mid \varphi \mathcal{R} \psi \mid \mathcal{H}_{H,I,O}\varphi \mid \mathcal{L}_{H,I,O}\varphi.$$

The *Leak* operator $\mathcal{L}$ is the dual of $\mathcal{H}$. Additionally, we introduce the common abbreviations $\mathsf{true} = v \vee \neg v$, $\mathsf{false} = \neg\mathsf{true}$, $\Diamond\varphi = \mathsf{true}\,\mathcal{U}\,\varphi$ and $\Box\varphi = \neg\Diamond\neg\varphi$.

Intuitively, $\mathcal{H}_{H,I,O}\varphi$ requires that the observable behavior of the system does not depend on the initial values of the *secret variables* $H$ in the desired time-frame, that is, before the formula $\varphi$ is satisfied. The operator specifies the power of the observer associated with it, by providing two sets of variables that are *visible to this observer*: a set $I$ of *input* variables and a set $O$ of *output* variables.

The Hide operator thus specifies what is to be considered the secret, what we consider to be observable, and when the secret may be released.

*Example 2.* We illustrate the use of SecLTL by providing examples of formal requirements for the clinical data management system from our running example.

The Hide operator allows us to specify precisely at which points some input variable is considered to be secret. If, for example, we are only interested in the first test result, then we can use the SecLTL formula $\mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}$ to express the requirement that the first test result has to remain secret forever. The observer in this scenario is specified to see the patient's consent status and the output variable $p$, which represents whether the system reports that some test in the past has been positive. He cannot observe the output variable $d$.

We can also restrict the traces on which a variable has to be kept secret, i.e, the traces on which a $\mathcal{H}$ formula needs to hold. The formula (1) below specifies

the property that only if the user never gives consent during the execution of the system, the first test result must never be revealed.

$$(\Box\neg c) \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}. \tag{1}$$

By nesting LTL operators and $\mathcal{H}$, we can place $\mathcal{H}$ in an appropriate temporal context and thus also refer to secrets introduced in multiple points of interest during the execution of the system. The following formula represents the property that every test result produced at a moment when the patient currently does not consent to the release of information must remain secret forever.

$$\Box\big(\neg c \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}\big). \tag{2}$$

A different policy for dealing with patient data might require that a test result is treated as confidential until (at some future point) the patient gives consent and that, at that future point in time, information about all test results from the past may be revealed in the system's public output $p$:

$$\Box\big(\mathcal{H}_{\{t\},\{t,c\},\{p\}}c\big). \tag{3}$$

The formulas (1), (2) and (3) above show that SecLTL allows formalizing different security policies, in particular ones that involve temporal requirements. In formula (2), the values of the input variable $t$ that are considered confidential depend on the temporal context. The temporal aspect of (3) is in the condition $c$, which determines from what point on *declassification* of the secret is allowed.

Since SecLTL is an extension of LTL, secrecy requirements can be combined with classical requirements on the system's behavior. For example, in addition to the first requirement above one can require that if the current test result is positive and the patient eventually consents in the future, then the existence of a positive test result is eventually reflected in the system's output. Formally:

$$\Box\big(t \wedge \Diamond c \rightarrow \Diamond p\big) \wedge \Box\big(\neg c \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}\big). \tag{4}$$

□

Although SecLTL specifications are linear-time properties, their semantics, more precisely the semantics of the Hide operator, is defined using a *set of alternative traces* and involves comparison of each of these traces to the *main trace*, i.e., the trace over which the SecLTL formula is interpreted.

**Definition 4 (Equivalences).** *Given a set of variables $V \subseteq \mathcal{V}_\mathcal{I} \cup \mathcal{V}_\mathcal{O}$, we define two elements $a$ and $a'$ of $\Sigma$, to be* observationally equivalent w.r.t. $V$*, noted $a \sim_V a'$, iff $a \cap V = a' \cap V$. Observational equivalence w.r.t. $V$ is extended to traces by pointwise comparison.*

**Definition 5 (Alternative traces).** *The* set of alternative traces *for an infinite trace $\pi \in \Sigma^\omega$ in $\mathcal{S}$ and a state $s \in S$ with respect to a set of secret variables $H \subseteq \mathcal{V}_\mathcal{I}$ and a set of input variables $I \subseteq \mathcal{V}_\mathcal{I}$ with $H \subseteq I$, is the set of traces*

*starting with a possibly different valuation of the variables $H$ in the first position, but otherwise adhering to the same values for the observable input variables $I$.*

$$\mathsf{Alt}_{\mathcal{S}}(s, \pi, H, I) = \{\ \pi' \in \mathsf{Traces}_{\mathcal{S},s} \mid \pi[0] \sim_{I \setminus H} \pi'[0] \text{ and } \pi[1,\infty) \sim_I \pi'[1,\infty)\ \}.$$

**Definition 6 (Semantics of SecLTL).** *Let $\mathcal{S} = (S, s^0, \mathcal{V}_{\mathcal{I}}, \mathcal{V}_{\mathcal{O}}, \Sigma, \delta)$ be a transition system and $s \in S$ be a state in $\mathcal{S}$. We say that the infinite trace $\pi \in \mathsf{Traces}_{\mathcal{S},s}$ and the state $s$ satisfy a given SecLTL formula $\varphi$, denoted $\mathcal{S}, s, \pi \models \varphi$ when the following conditions are satisfied:*

- *For an atomic proposition, i.e, a variable $p \in \mathcal{V}_{\mathcal{I}} \cup \mathcal{V}_{\mathcal{O}}$:*

$$\mathcal{S}, s, \pi \models p \text{ iff } p \in \pi[0].$$

- *For the boolean connectives:*

$$\begin{array}{ll} \mathcal{S}, s, \pi \models \neg\psi & \text{iff } \mathcal{S}, s, \pi \not\models \psi, \\ \mathcal{S}, s, \pi \models \varphi_1 \vee \varphi_2 & \text{iff } \mathcal{S}, s, \pi \models \varphi_1 \text{ or } \mathcal{S}, s, \pi \models \varphi_2, \\ \mathcal{S}, s, \pi \models \varphi_1 \wedge \varphi_2 & \text{iff } \mathcal{S}, s, \pi \models \varphi_1 \text{ and } \mathcal{S}, s, \pi \models \varphi_2. \end{array}$$

- *For classical temporal operators, where $\sigma = \mathsf{Exec}_{\mathcal{S}}(s, \pi)$:*

$$\begin{array}{ll} \mathcal{S}, s, \pi \models \bigcirc\psi & \text{iff } \mathcal{S}, \sigma[1], \pi[1,\infty) \models \psi, \\ \mathcal{S}, s, \pi \models \varphi_1 \mathcal{U} \varphi_2 & \text{iff for some } i \geq 0, \text{ we have } \mathcal{S}, \sigma[i], \pi[i,\infty) \models \varphi_2 \\ & \quad \text{and for all } 0 \leq j < i \text{ we have } \mathcal{S}, \sigma[j], \pi[j,\infty) \models \varphi_1, \\ \mathcal{S}, s, \pi \models \varphi_1 \mathcal{R} \varphi_2 & \text{iff for all } i \geq 0, \text{ we have } \mathcal{S}, \sigma[i], \pi[i,\infty) \models \varphi_2, \text{ or} \\ & \quad \text{for some } i \geq 0, \mathcal{S}, \sigma[i], \pi[i,\infty) \models \varphi_1 \text{ and} \\ & \quad \text{for all } 0 \leq j \leq i \text{ we have } \mathcal{S}, \sigma[j], \pi[j,\infty) \models \varphi_2. \end{array}$$

- *For the modal operators $\mathcal{H}$ and $\mathcal{L}$, where $\sigma = \mathsf{Exec}_{\mathcal{S}}(s, \pi)$:*

$$\begin{array}{ll} \mathcal{S}, s, \pi \models \mathcal{H}_{H,I,O}\psi & \text{iff for every } \pi' \in \mathsf{Alt}_{\mathcal{S}}(s, \pi, H, I) \text{ we have } \pi \sim_O \pi', \\ & \quad \text{or for some } i \geq 0 \text{ we have } \mathcal{S}, \sigma[i], \pi[i,\infty) \models \psi \\ & \quad \text{and } \pi[0,i) \sim_O \pi'[0,i) \text{ for every } \pi' \in \mathsf{Alt}_{\mathcal{S}}(s, \pi, H, I), \\ \mathcal{S}, s, \pi \models \mathcal{L}_{H,I,O}\psi & \text{iff for some } \pi' \in \mathsf{Alt}_{\mathcal{S}}(s, \pi, H, I) \text{ and } i \geq 0, \pi[i] \not\sim_O \pi'[i] \\ & \quad \text{and for all } 0 \leq j \leq i, \mathcal{S}, \sigma[j], \pi[j,\infty) \models \psi. \end{array}$$

*Remark 1.* Note that the secret specified by each (semantic) occurrence of the Hide operator in a SecLTL formula consists of the individual valuation of the variables in the set $H$ at the current point of the trace. Thus, for example, the formula $\mathcal{H}_{\{h\},\{h\},\{o\}}\mathsf{false} \wedge \bigcirc\mathcal{H}_{\{h\},\{h\},\{o\}}\mathsf{false}$ specifies that the first value of $h$ must be secret forever and the second value of $h$ must be secret forever. Thus, on a trace that satisfies this formula each of the inputs is kept secret *individually*, but some *correlation* between them might never the less be revealed.

*Example 3.* Let us consider again the SecLTL formulas (1), (2), (3) and (4) and the transition systems from Figures 1(a) and 1(b). The formula (1) is satisfied on every trace allowed by $\mathcal{S}_A$, because on all traces where the variable $c$ never becomes *true*, the value of $p$ is also always *false*. Since the alternative traces

defined by $\mathcal{H}_{\{t\},\{t,c\},\{p\}}$ agree with the main trace on the values of $c$, the same holds for them, and, hence, they agree with the main trace on the value of $p$.

To see that formula (2) does not hold for some trace allowed by $\mathcal{S}_A$, consider the infinite trace $\pi_1 = c\bar{t}\bar{p}\bar{d}, \bar{c}\bar{t}\bar{p}\bar{d}, \bar{c}t\bar{p}d, c\bar{t}pd, (c\bar{t}pd)^\omega$, whose corresponding execution from the initial state $s_1$ in $S_A$ is $\mathsf{Exec}_{\mathcal{S}_A}(s_1, \pi_1) = s_1, s_3, s_1, s_2, s_4, s_4^\omega$. The formula $(\neg c \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}})$ is violated at position 2, since $\pi_1[2,\infty) \models \neg c$ and $\pi_1[2,\infty) \not\models \mathcal{H}_{\{t\},\{t,c\},\{p\}}$, since there exists an alternative trace, $\pi_1' \in \mathsf{Alt}_{\mathcal{S}}(s_1, \pi_1[2,\infty), \{t\}, \{t,c\})$ on which the system's output is different. Such a trace is $\pi_1' = \bar{c}\bar{t}\bar{p}\bar{d}, c\bar{t}\bar{p}\bar{d}$, with corresponding execution $\mathsf{Exec}_{\mathcal{S}_A}(s_1, \pi_1') = s_1, s_1, s_3$.

The formula (3) is clearly satisfied on each possible trace allowed by $\mathcal{S}_A$, because for each position where $p$ is satisfied, the variable $c$ is *true* as well.

Since the formula (2) is one of the conjuncts in formula (4), formula (4) is also not satisfied by the trace $\pi_1$. Note, however that the other conjunct, i.e., $\Box\big(t \wedge \Diamond c \rightarrow \Diamond p\big)$ is satisfied by every trace in $\mathsf{Traces}_{S_A,s_1}$, because for all executions on which the left hand side of the implication is true visit state $s_4$ (and hence $p$ is eventually true on the corresponding trace).

Using the same arguments as above, one can see that the formulas (1) and (3) are satisfied by each trace in $\mathsf{Traces}_{\mathcal{S}_B,s_1}$ as well.

In the system $\mathcal{S}_B$, the trace $\pi_1$ is not possible, i.e., $\pi_1 \notin \mathsf{Traces}_{\mathcal{S}_B,s_1}$, as we saw earlier (looking at a finite prefix of $\pi_1$). While formula (2) is satisfied by each trace in $\mathsf{Traces}_{\mathcal{S}_B,s_1}$, for (4) there exists a counterexample trace, because all information about the existence of a positive test in the past is lost when the edge from state $s_2$ to state $s_3$ is taken.

### 2.3 Finite-trace Semantics

For runtime monitoring we must interpret temporal formulas on *finite* traces.

In the case of SecLTL, we first have to adapt the definition of alternative traces. For a finite trace $\pi \in \Sigma^*$, state $s \in S$, and sets of variables $H, I \subseteq \mathcal{V}_\mathcal{I}$ with $H \subseteq I$, we define the set of *finite* alternative traces as follows:

$$\mathsf{AltFin}_{\mathcal{S}}(s, \pi, H, I) = \{\, \pi' \in \mathsf{TracesFin}_{\mathcal{S},s} \mid |\pi'| = |\pi|, \pi[0] \sim_{I\setminus H} \pi'[0] \text{ and } \pi[1,|\pi|) \sim_I \pi'[1,|\pi'|) \,\}.$$

We can again define inductively $\mathcal{S}, s, \pi \models \varphi$ for a finite trace $\pi$, state $s$ and a SecLTL formula $\varphi$. The interpretation of propositional variables and boolean operators remains unchanged. The interpretation of the classical LTL operators coincides with the standard finite-trace interpretation[1]. For the $\mathcal{H}$ operator, the finite-trace semantics is similar to that of the LTL weak until ($\mathcal{W}$) operator.

– For classical temporal operators, where $|\pi| = n$ and $\sigma = \mathsf{Exec}_{\mathcal{S}}(s, \pi)$:

[1] We use a simple two-valued finite-trace semantics, as described for example in [8]. For an overview on other finite-trace semantics used in runtime verification, we refer the reader to [9].

$$
\begin{array}{lll}
\mathcal{S}, s, \pi \models \bigcirc \psi & \text{iff} & n > 1 \text{ and } \mathcal{S}, \sigma[1], \pi[1, n) \models \psi, \\
\mathcal{S}, s, \pi \models \varphi_1 \mathcal{U} \varphi_2 & \text{iff} & \text{for some } 0 \leq i < n, \text{ we have } \mathcal{S}, \sigma[i], \pi[i, n) \models \varphi_2 \\
 & & \text{and for all } 0 \leq j < i \text{ we have } \mathcal{S}, \sigma[j], \pi[j, n) \models \varphi_1, \\
\mathcal{S}, s, \pi \models \varphi_1 \mathcal{R} \varphi_2 & \text{iff} & \text{for all } 0 \leq i < n, \text{ we have } \mathcal{S}, \sigma[i], \pi[i, n) \models \varphi_2, \text{ or} \\
 & & \text{for some } 0 \leq i < n, \mathcal{S}, \sigma[i], \pi[i, n) \models \varphi_1 \text{ and} \\
 & & \text{for all } 0 \leq j \leq i \text{ we have } \mathcal{S}, \sigma[j], \pi[j, n) \models \varphi_2.
\end{array}
$$

– For the modal operators $\mathcal{H}$ and $\mathcal{L}$, where $|\pi| = n$ and $\sigma = \mathsf{Exec}_{\mathcal{S}}(s, \pi)$:

$$
\begin{array}{ll}
\mathcal{S}, s, \pi \models \mathcal{H}_{H,I,O} \psi & \text{iff } \pi \sim_O \pi' \text{ for every } \pi' \in \mathsf{AltFin}_{\mathcal{S}}(s, \pi, H, I), \text{ or} \\
 & \text{for some } 0 \leq i < n, \mathcal{S}, \sigma[i], \pi[i, n) \models \psi \text{ and for all} \\
 & \pi' \in \mathsf{AltFin}_{\mathcal{S}}(s, \pi, H, I), \pi[0, i) \sim_O \pi'[0, i), \\
\mathcal{S}, s, \pi \models \mathcal{L}_{H,I,O} \psi & \text{iff for some } \pi' \in \mathsf{AltFin}_{\mathcal{S}}(s, \pi, H, I) \text{ and } 0 \leq i < n, \\
 & \pi[i] \not\sim_O \pi'[i] \text{ and for all } 0 \leq j \leq i, \mathcal{S}, \sigma[j], \pi[j, n) \models \psi.
\end{array}
$$

*Example 4.* According to the finite trace semantics, formulas (1) and (3) are satisfied by all traces in $\mathsf{TracesFin}_{\mathcal{S}_A, s_1}$ and $\mathsf{TracesFin}_{\mathcal{S}_B, s_1}$. The argument for (3) from before directly applies, because the only temporal operators that occur in it are $\Box$ and $\mathcal{H}$. For (1), the argument is that the formula $\mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}$ is satisfied on every finite trace on which the formula $\Diamond c$ does not hold (according to our finite trace semantics, if we have not seen a $c$ yet, $\Diamond c$ is not satisfied).

For formula (2), a trace in $\mathsf{TracesFin}_{\mathcal{S}_A, s_1}$ for which the formula does not hold is obtained by taking the finite prefix $\pi_1[0, 3]$ of the infinite counterexample trace $\pi_1$ considered earlier. Again, (2) is satisfied by each trace in $\mathsf{TracesFin}_{\mathcal{S}_B, s_1}$.

## 3 Monitoring SecLTL

Our monitoring algorithm is based on a translation of the SecLTL specification and the transition system into an alternating automaton, which we call the *monitoring automaton*. The monitoring automaton keeps track of the temporal specification and ensures the observational equivalence of the alternative traces. In the following two subsections, we first describe the construction of the automaton and then define the monitoring algorithm, which constructs the possible run trees of the monitoring automaton on-the-fly while reading the trace.

### 3.1 From SecLTL Formulas to Automata

We now describe a translation from SecLTL formulas and transition systems to alternating automata, applying the finite-trace semantics defined in Section 2.3. A similar construction for the infinite-trace semantics is given in [6].

**Definition 7 (Alternating automaton).** *An* alternating automaton *is a tuple* $\mathcal{A} = (Q, q_0, \Sigma, \rho, F)$*, where*

– $Q$ *is a finite set of states and* $q_0 \in Q$ *is the initial state,*
– $\Sigma$ *is the finite alphabet of the automaton,*

– $\rho : Q \times \Sigma \to \mathcal{B}^+(Q)$ *is a transition function that maps a state in* $Q$ *and a letter from* $\Sigma$ *to a positive boolean combination of states, i.e., formulas built from the formulas* true, false *and the elements of* $Q$ *using* $\wedge$ *and* $\vee$,
– $F \subseteq Q$ *is the set of accepting states.*

The run of an alternating automaton $\mathcal{A}$ is in general a tree. A finite $Q$-labeled tree $(T, r)$ for a finite set $Q$ consists of a finite tree $T$ and a labelling function $r : T \to Q$ which labels every node of $T$ with an element of $Q$. The tree $T$ can be represented as a finite subset of $\mathbb{N}^*_{>0}$, where each node $\tau$ in the tree is a sequence of positive integers and for every $\tau \in \mathbb{N}^*_{>0}$ and $n \in \mathbb{N}_{>0}$, if $\tau \cdot n \in T$ then:

– $\tau \in T$ (i.e., $T$ is prefix-closed) and there is an edge from $\tau$ to $\tau \cdot n$, and
– for every $m \in \mathbb{N}^*_{>0}$ with $m < n$ it holds that $\tau \cdot m \in T$.

The root of $T$ is the empty sequence $\epsilon$ and for a node $\tau \in T$, $|\tau|$ is the distance of the node $\tau$ from the root of the tree.

**Definition 8 (Run).** *A* run *of an alternating automaton* $\mathcal{A} = (Q, q_0, \Sigma, \rho, F)$ *on a finite word* $\pi \in \Sigma^*$ *is a finite* $Q$*-labeled tree* $(T, r)$ *such that:*

– $r(\epsilon) = q_0$*, that is, the root of the tree is labeled with the initial state, and*
– *for every node* $\tau$ *in* $T$ *with children* $\tau_1, \ldots, \tau_k$ *it holds that* $k \leq |Q|$ *and if* $q = r(\tau)$ *is the label of* $\tau$ *and* $i = |\tau|$ *is its distance from the root, then the set of labels of its children* $\{r(\tau_1), \ldots, r(\tau_k)\}$ *satisfies the formula* $\rho(q, \pi[i])$.

**Definition 9 (Language).** *A run of* $\mathcal{A}$ *on a finite word* $\pi$ *is* accepting *if every path through the tree ends in an accepting state. A finite word* $\pi$ *is* accepted *by* $\mathcal{A}$ *if there exists an accepting run of* $\pi$ *in* $\mathcal{A}$*. We denote the* language of $\mathcal{A}$*, that is, the set of finite sequences accepted by* $\mathcal{A}$*, by* $\mathcal{L}^*(\mathcal{A})$.

Let $\mathcal{S} = (S, s^0, \mathcal{V}_\mathcal{I}, \mathcal{V}_\mathcal{O}, \Sigma, \delta)$ be a transition system and $\varphi$ be a SecLTL formula over the set of input variables $\mathcal{V}_\mathcal{I}$ and the set of output variables $\mathcal{V}_\mathcal{O}$. We can assume that all negations in the formula have been pushed to the level of the atomic propositions. For propositional and classical LTL operators this can be achieved using standard rewrite rules and for the $\mathcal{H}$ operator using $\mathcal{L}$ [6].

The alternating automaton $\mathcal{A}_\mathcal{S}(\varphi) = (Q, q_0, \Sigma, \rho, F)$ for the transition system $\mathcal{S}$ and the SecLTL formula $\varphi$ is defined as follows.

The set of states $Q$ consists of states corresponding to the subformulas of $\varphi$ together with special states corresponding to the $\mathcal{H}$ and $\mathcal{L}$ subformulas of $\varphi$:

$$\begin{aligned} Q = {} & \{\ (\psi, s) \mid \psi \text{ is a subformula of } \varphi \text{ and } s \in S\ \} \cup \{\ \mathit{accept}\ \} \\ & \cup \{\ ((\tilde{s}, I, O, \mathcal{H}, \psi), s) \mid \tilde{s}, s \in S \text{ and } \exists H.\ \mathcal{H}_{H,I,O}\psi \text{ is subformula of } \varphi\ \} \\ & \cup \{\ ((\tilde{s}, I, O, \mathcal{L}, \psi), s) \mid \tilde{s}, s \in S \text{ and } \exists H.\ \mathcal{L}_{H,I,O}\psi \text{ is subformula of } \varphi\ \}. \end{aligned}$$

The initial state of $\mathcal{A}_\mathcal{S}(\varphi)$ is $q_0 = (\varphi, s^0)$, where $s^0$ is the initial state of $\mathcal{S}$. The set $F$ of accepting states, that contains the state *accept*, is defined as

$$\begin{aligned} F = {} & \{\ \mathit{accept}\ \} \cup \{\ (\varphi_1 \mathcal{R} \varphi_2, s) \in Q\ \} \cup \\ & \{\ (\mathcal{H}_{H,I,O}\psi, s) \in Q\} \cup \{\ ((\tilde{s}, I, O, \mathcal{H}, \psi), s) \in Q\ \}. \end{aligned}$$

The transition function $\rho$ of the automaton is defined recursively as follows.

For $s \in S$ and $a \in \Sigma$ such that $\delta(s,a)$ is undefined, we define $\rho((\psi,s),a) = \mathsf{false}$, $\rho(((\tilde{s},I,O,\mathcal{H},\psi),s),a) = \mathsf{false}$, $\rho(((\tilde{s},I,O,\mathcal{L},\psi),s),a) = \mathsf{false}$.

Below we consider only the cases when $\delta(s,a)$ is defined.

For an atomic proposition $p \in \mathcal{V}_\mathcal{I} \cup \mathcal{V}_\mathcal{O}$:

$$\rho((p,s),a) = \mathit{accept} \text{ if } p \in a \text{ and } \rho((p,s),a) = \mathsf{false} \text{ otherwise,}$$
$$\rho((\neg p,s),a) = \mathit{accept} \text{ if } p \notin a \text{ and } \rho((\neg p,s),a) = \mathsf{false} \text{ otherwise.}$$

For SecLTL formulas $\varphi_1$, $\varphi_2$ and $\psi$:

$$\begin{array}{lcl}
\rho((\varphi_1 \wedge \varphi_2, s), a) & = & \rho((\varphi_1, s), a) \wedge \rho((\varphi_2, s), a), \\
\rho((\varphi_1 \vee \varphi_2, s), a) & = & \rho((\varphi_1, s), a) \vee \rho((\varphi_2, s), a), \\
\rho((\bigcirc \psi, s), a) & = & (\psi, \delta(s,a)), \\
\rho((\varphi_1 \mathcal{U} \varphi_2, s), a) & = & \rho((\varphi_2, s), a) \vee \big(\rho((\varphi_1, s), a) \wedge (\varphi_1 \mathcal{U} \varphi_2, \delta(s,a))\big), \\
\rho((\varphi_1 \mathcal{R} \varphi_2, s), a) & = & \rho((\varphi_2, s), a) \wedge \big(\rho((\varphi_1, s), a) \vee (\varphi_1 \mathcal{R} \varphi_2, \delta(s,a))\big).
\end{array}$$

For SecLTL formula $\psi$ and sets $H, I \subseteq \mathcal{V}_\mathcal{I}$ and $O \subseteq \mathcal{V}_\mathcal{O}$:

$$\begin{array}{lcl}
\rho((\mathcal{H}_{H,I,O}\psi, s), a) & = & \rho((\psi, s), a) \vee \big(\mathsf{check}(O, a, \mathsf{Alt}_\Sigma(s,a,H,I)) \wedge \\
 & & \bigwedge_{\tilde{a} \in \mathsf{Alt}_\Sigma(s,a,H,I)} ((\delta(s,\tilde{a}), I, O, \mathcal{H}, \psi), \delta(s,a))\big), \\
\rho((\mathcal{L}_{H,I,O}\psi, s), a) & = & \rho((\psi, s), a) \wedge \big(\neg\mathsf{check}(O, a, \mathsf{Alt}_\Sigma(s,a,H,I)) \vee \\
 & & \bigvee_{\tilde{a} \in \mathsf{Alt}_\Sigma(s,a,H,I)} ((\delta(s,\tilde{a}), I, O, \mathcal{L}, \psi), \delta(s,a))\big),
\end{array}$$

where for $s \in S$, $a \in \Sigma$, and $H, I \subseteq \mathcal{V}_\mathcal{I}$ and $O \subseteq \mathcal{V}_\mathcal{O}$ we define:

$$\mathsf{Alt}_\Sigma(s,a,H,I) = \{\tilde{a} \in \Sigma \mid \tilde{a} \sim_{I \setminus H} a \text{ and } \exists s' \in S. s' = \delta(s,\tilde{a})\},$$

$$\mathsf{check}(O,a,A) = (\forall \tilde{a} \in A : \tilde{a} \sim_O a).$$

For $((\tilde{s},I,O,\mathcal{H},\psi),s) \in Q$ and $((\tilde{s},I,O,\mathcal{L},\psi),s) \in Q$ we define:

$$\begin{array}{lcl}
\rho(((\tilde{s}, I, O, \mathcal{H}, \psi), s), a) & = & \rho((\psi, s), a) \vee \big(\mathsf{check}(O, a, \mathsf{Alt}_\Sigma(\tilde{s}, a, \emptyset, I)) \wedge \\
 & & \bigwedge_{\tilde{a} \in \mathsf{Alt}_\Sigma(\tilde{s},a,\emptyset,I)} ((\delta(\tilde{s},\tilde{a}), I, O, \mathcal{H}, \psi), \delta(s,a))\big), \\
\rho(((\tilde{s}, I, O, \mathcal{L}, \psi), s), a) & = & \rho((\psi, s), a) \wedge \big(\neg\mathsf{check}(O, a, \mathsf{Alt}_\Sigma(\tilde{s}, a, \emptyset, I)) \vee \\
 & & \bigvee_{\tilde{a} \in \mathsf{Alt}_\Sigma(\tilde{s},a,\emptyset,I)} ((\delta(\tilde{s},\tilde{a}), I, O, \mathcal{L}, \psi), \delta(s,a))\big).
\end{array}$$

Finally, we define $\rho(\mathit{accept}, a) = \mathit{accept}$.

**Definition 10 (Monitor automaton).** *Given a transition system $\mathcal{S} = (S, s^0, \mathcal{V}_\mathcal{I}, \mathcal{V}_\mathcal{O}, \Sigma, \delta)$ and a SecLTL formula $\varphi$, the* monitor automaton *for $\varphi$ is the automaton $\mathcal{A}_\mathcal{S}(\varphi)$ defined above.*

The monitor automaton $\mathcal{A}_\mathcal{S}(\varphi)$ has the property that for every finite trace $\pi \in \Sigma^*$, it holds that $\pi \in \mathcal{L}^*(\mathcal{A}_\mathcal{S}(\varphi))$ iff $\pi \in \mathsf{TracesFin}_{\mathcal{S},s^0}$ and $\mathcal{S}, s^0, \pi \models \varphi$.

*Example 5.* We now give the alternating automaton $\mathcal{A}_{\mathcal{S}_A}(\varphi)$ for the SecLTL property $\varphi = (\Box\neg c) \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}$ and the transition system $\mathcal{S}_A$. The set of states and the transition relation of $\mathcal{A}_{\mathcal{S}_A}(\varphi)$ are given in Figure 2.

| label | state | label | state |
|---|---|---|---|
| $q_0$ | $((\Box\neg c) \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}, s_1)$ | *accept* | *accept* |
| $q_1$ | $(\Diamond c, s_1)$ | $q_2$ | $(\Diamond c, s_2)$ |
| $q_3$ | $((s_1, \{t,c\}, \{p\}, \mathcal{H}, \mathsf{false}), s_1)$ | $q_4$ | $((s_2, \{t,c\}, \{p\}, \mathcal{H}, \mathsf{false}), s_1)$ |
| $q_5$ | $((s_1, \{t,c\}, \{p\}, \mathcal{H}, \mathsf{false}), s_2)$ | $q_6$ | $((s_2, \{t,c\}, \{p\}, \mathcal{H}, \mathsf{false}), s_2)$ |
| $q_7$ | $((s_3, \{t,c\}, \{p\}, \mathcal{H}, \mathsf{false}), s_3)$ | $q_8$ | $((s_4, \{t,c\}, \{p\}, \mathcal{H}, \mathsf{false}), s_4)$ |

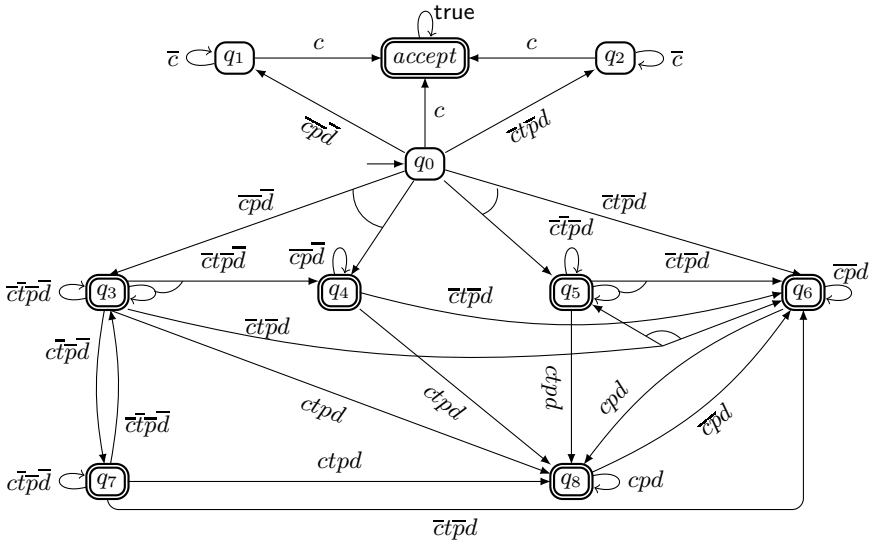



**Fig. 2.** The alternating automaton $\mathcal{A}_{\mathcal{S}_A}(\varphi)$ for the transition system $\mathcal{S}_A$ from Figure 1(a) and the SecLTL formula $\varphi = (\Box\neg c) \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}$. Branchings with arcs represent conjunctions, branchings without arcs are to be interpreted disjunctively. The states drawn with double line are the accepting states of $\mathcal{A}_{\mathcal{S}_A}(\varphi)$.

In the initial state the automaton can either decide to refute the left side of the implication (by waiting for a $c$ in state $t_1$ or state $t_2$) or it has to validate the hide operator in which case it has to check whether the corresponding pairs of main and alternative traces in $\mathcal{S}_A$ are observationally equivalent w.r.t. the output variable $p$. The equivalence check is integrated in the transition relation.

### 3.2 The Monitoring Algorithm

Trace checking algorithms for alternating automata attempt to construct an accepting run tree. Different traversal strategies, such as depth-first, breadth-first, or bottom-up, result in trace checking algorithms with different performance characteristics [8]. For monitoring, where the trace becomes available incrementally, a good strategy is to construct the run tree in a breadth-first manner. Conceptually, the monitoring algorithm maintains a set of candidate trees and adds a new layer at the leaves whenever a new position in the trace becomes available. However, since neither the construction of the next layer nor the verification of acceptance condition refer to any non-leaf nodes of the tree, it in fact suffices to keep track of the states on the leaves. The state of the monitor is

```
MONITOR-SECLTL(S, φ, π)
  (Q, q0, Σ, ρ, F) ← A_S(φ)
  D ← {{q0}}
  for n = 0 to |π| − 1 do
    D' ← ∅
    for each C ∈ D do
      D' ← D' ∪ successors(C, π[n])
    end for
    D ← D'
  end for
  return ACCEPT(D, F)

ACCEPT(D, F)
  D' ← ∅
  for each C ∈ D do
    if accepting(C, F) then
      D' ← D' ∪ {C}
    end if
  end for
  return (D' ≠ ∅)
```

**Fig. 3.** Monitoring algorithm for a transition system $\mathcal{S}$, a SecLTL formula $\varphi$, and a finite trace $\pi \in \mathsf{TracesFin}_{\mathcal{S},s^0}$. The algorithm returns *true* iff $\mathcal{S}, s^0, \pi \models \varphi$.

therefore represented by a set $D$ of sets $C$ of states, where each set $C$ corresponds to the states on the leaves of some partially constructed run tree. For a more detailed explanation of the breadth-first strategy, we refer the reader to [8].

The monitoring algorithm shown in Figure 3 applies the breadth-first strategy to the monitoring automaton defined in the previous section. Initially, there is only one candidate tree, consisting of a single node labeled with the initial state $q_0$ of $\mathcal{A}_\mathcal{S}(\varphi)$. Variable $D$ is therefore initialized with a singleton set containing the singleton set which consists of $q_0$. For each position of the trace, the successor sets of the elements $C \in D$ are computed by the *successors* function, where $\mathit{successors}(C, a) = \bigotimes_{q \in C} \mathit{next}(\rho(q, a))$, $\otimes$ denotes the crossproduct $\{C_1, \ldots, C_n\} \otimes \{C'_1, \ldots, C'_m\} = \{C_i \cup C'_j \mid i = 1 \ldots n, j = 1 \ldots m\}$, and function *next* computes the set of sets of successors defined by the positive Boolean combination in the transition function as follows:

$$\begin{aligned} \mathit{next}(q) &= \{\{q\}\} \text{ for } q \in Q, \\ \mathit{next}(\theta_1 \wedge \theta_2) &= \mathit{next}(\theta_1) \otimes \mathit{next}(\theta_2), \\ \mathit{next}(\theta_1 \vee \theta_2) &= \mathit{next}(\theta_1) \cup \mathit{next}(\theta_2). \end{aligned}$$

At any point, we can check if there exists a run tree for the trace seen so far, by searching for an element $C$ of $D$ that consists entirely of accepting states. In the algorithm shown in Figure 3, it is assumed that we are only interested in the result at the end of the trace, after $|\pi|$ steps. Function *accepting* checks if all states are accepting, and the algorithm keeps only those elements of $D$ that satisfy this check. If the resulting set $D'$ is non-empty, we know that there exists a run tree, and the algorithm returns *true*.

*Example 6.* We monitor the SecLTL formula $(\Box\neg c) \rightarrow \mathcal{H}_{\{t\},\{t,c\},\{p\}}\mathsf{false}$ and the transition system $\mathcal{S}_A$ from Figure 1(a) on prefixes of the trace $\pi$ shown below using the monitoring automaton depicted in Figure 2. The row marked "result" indicates in the $i$th column the monitoring result obtained after monitoring the prefix $\pi[0, i)$ of $\pi$ consisting of the first $i$ positions.

| step | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $\pi$ : | $\overline{c}t\overline{p}d$ | $\overline{c}\overline{t}\overline{p}d$ | $c\overline{t}pd$ | $\overline{c}t\overline{p}d$ |
| $D$ | $\{\{q_1\},\{q_3,q_4\},$ $\{q_2\},\{q_5,q_6\}\}$ | $\{\{q_1\},$ $\{q_2\},\{q_5,q_6\}\}$ | $\{\{accept\},$ $\{accept\}\}$ | $\{\{accept\},$ $\{accept\}\}$ |
| result | *true* | *true* | *true* | *true* |

### 3.3 Towards Stronger Security Guarantees

Previous approaches to monitoring information flow [1, 4, 5] only consider *sequential* programs that read all their inputs at the beginning of the execution, and thus the secrets are only introduced at the initial state. For this case, monitoring the SecLTL property $\mathcal{H}_{H,I,O}\mathsf{false}$ using the algorithm from Section 3.2 provides the same security guarantees as these approaches.

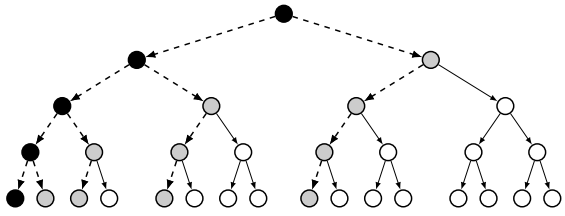
However, SecLTL allows for specifying more complex information flow properties for reactive programs - in particular ones that refer to multiple secrets, which may be introduced at different points of time. A prominent example is noninterference [2] for reactive systems. A reactive system is noninterferent, if for any two executions that have indistinguishable sequences of inputs the observer cannot distinguish the sequences of outputs. We can characterize noninterference by the SecLTL formula $\varphi_{ni} = \Box\mathcal{H}_{H,I,O}\mathsf{false}$ , where $H$ is the input that must be hidden from the observer and $I$ and $O$ are the input and output revealed to him. A system satisfies noninterference if and only if $\varphi_{ni}$ holds on *all traces* of the system. Thus, when monitoring noninterference we must verify $\varphi_{ni}$ along more than a single trace in order to be sure to detect every violation.

The SecLTL semantics guarantees that after successfully monitoring a single trace, none of the secrets specified in the formula is revealed. However, this does not exclude disclosure of correlations between different secrets.

Consider the program shown in Figure 4(a), which reads, in each iteration of the loop, a binary input and it outputs whether the sum over the input bits seen so far exceeds 1. The transition system generated by the program, which is shown in Figure 4(b), does not satisfy noninterference, because an observer

```
int x=0;
while true do
   x = x + input();
   output(x> 1);
end while
```

(a)



(b)

**Fig. 4.** (a) Program that reads, in each iteration of its loop, a binary input and outputs whether the sum over the input bits seen so far exceeds 1. (b) The corresponding transition system branches to the left on input 0 and to the right on input 1. The dashed arrows indicate output 0, the solid arrows output 1. The black nodes identify the execution corresponding to the monitored trace, with constant input 0. The gray nodes identify the paths corresponding to the alternative traces.

cannot draw a distinction between the two streams of inputs corresponding to, for example, the left-most and the right-most trace, but the same observer can certainly draw a distinction between the streams of outputs. However, if we monitor $\varphi_{ni}$ only on the left-most trace with constant input 0 (shown with black nodes in Figure 4(b)), we will not detect this violation, because the alternative traces (depicted with gray nodes) produce the same sequence of outputs. In order to detect the violation, we must monitor at least one additional trace.

A possible solution would be to monitor, in addition to the given trace, the set of traces that have the same history of observable inputs. Clearly, the efficiency of such an algorithm depends on the representation of the resulting monitor state, and there is room for optimizations and heuristics. We discuss some ideas for future work in this direction in the following section.

## 4 Outlook and Conclusions

SecLTL is an attractive specification language for security-critical reactive systems, because it allows us to state precisely *when* and *under what conditions* an event must remain secret. For large systems, where SecLTL model checking [6] might be too expensive, the monitoring approach presented in this paper provides a much more practical alternative. Monitoring is dramatically cheaper than model checking, because the on-the-fly construction only explores that part of the system that corresponds to the observed trace and its alternatives as defined by the SecLTL specification. In general, however, the monitor may also need to traverse a substantial part of the system's state space. In this case, the explicit state representation of our monitoring algorithm is a limitation, and an important direction for future work is to integrate symbolic state representations and abstraction techniques from software model checking and abstract interpretation into the monitoring algorithm.

Another research direction concerns the extension of the monitoring algorithm towards more general security guarantees as discussed in Section 3.3. Monitoring sets of traces, as suggested there, may turn out to be too expensive if variations in the secrets force the monitor to explore a significant portion of the system state space in parallel. In practice, it may be possible to trade some loss of precision for a substantial gain in efficiency. In a probabilistic setting, for example, it might be possible to select a small set of traces that guarantee a reasonable limit on the loss of entropy from the observer's point of view.

*Acknowledgements.* This work was partially supported by the German Research Foundation (DFG) under the project SpAGAT (grant no. FI 936/2-1) in the priority program "Reliably Secure Software Systems – RS3".

## References

1. Sabelfeld, A., Russo, A.: From dynamic to static and back: riding the roller coaster of information-flow control research. In: Proceedings of the 7th international An-

drei Ershov Memorial conference on Perspectives of Systems Informatics. PSI'09, Springer-Verlag (2010) 352–365
2. Goguen, J.A., Meseguer, J.: Security policies and security models. In: Proceedings of S&P. (1982) 11–20
3. Department of Health, The Caldicott Committee, Caldicott, F.: Report on the review of patient-identifiable information. Department of Health, London (1997)
4. Russo, A., Sabelfeld, A.: Dynamic vs. static flow-sensitive security analysis. In: Proc. CSF'10, IEEE Computer Society (2010) 186–199
5. Guernic, G.L., Banerjee, A., Jensen, T., Schmidt, D.A.: Automata-based confidentiality monitoring. In: ASIAN06: 11th Asian Computing Science Conference on Secure Software. (2006)
6. Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M.N., Seidl, H.: Model checking information flow in reactive systems. In: Proceedings of VMCAI. (2012) 169–185
7. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In Moller, F., Birtwistle, G., eds.: Logics for Concurrency. Structure versus Automata. Volume 1043 of Lecture Notes in Computer Science., Springer-Verlag (1996) 238–266
8. Finkbeiner, B., Sipma, H.: Checking finite traces using alternating automata. Form. Methods Syst. Des. **24** (2004) 101–127
9. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Journal of Logic and Computation **20** (2010) 651–674