Information Flow for Timed Automata

Flemming Nielson(✉), Hanne Riis Nielson, and Panagiotis Vasilikos

Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Kongens Lyngby, Denmark
{fnie,hrni,panva}@dtu.dk

Abstract. One of the key demands of cyberphysical systems is that they meet their safety goals. *Timed Automata* has established itself as a formalism for modelling and analysing the real-time safety aspects of cyberphysical systems. Increasingly it is also demanded that cyberphysical systems meet a number of security goals for confidentiality and integrity. *Information Flow Control* is an approach to ensuring that there are no flows of information that violate the stated security policy.

We develop a language based approach to the modelling and analysis of timed systems that allows to incorporate considerations of information flow control. We define a type system for information flow that takes account of the non-determinism and clocks of timed systems. The adequacy of the type system is ensured by means of a non-interference result.

1 Introduction

Motivation. Embedded systems are key components of cyberphysical systems and are often subject to stringent safety goals. Among the current approaches to the modelling and analysis of timed systems, the approach of *Timed Automata* [3] stands out as being a very successful approach with well-developed tool support – in particular the *UPPAAL* suite [16] of tools.
As cyberphysical systems become increasingly distributed and interconnected through wireless communication links it becomes even more important to ensure that they meet suitable security goals. This may involve safeguarding the confidentiality (or privacy) of sensor data or ensuring the integrity (or authenticity) of control commands; in both cases we need to limit the way information flows through the program. *Information Flow Control* [9,17] is a key approach to ensuring that software systems admit no flow of information that violate the stated security policy for confidentiality and/or integrity.

Contribution. It is therefore natural to extend the enforcement of safety properties of Timed Automata with the enforcement of appropriate Information Flow policies. It is immediate that the treatment of *clocks* will pose a challenge. It turns out that the *non-determinism* inherent in automata poses another challenge. More fundamentally there is the challenge that Timed Automata is an

L. Aceto et al. (Eds.): Larsen Festschrift, LNCS 10460, pp. 3–21, 2017.
DOI: 10.1007/978-3-319-63121-9_1

automata based formalism whereas most approaches to Information Flow take a language based approach by developing type systems for programming languages or process calculi.

Consequently we take a language based approach to the study of timed systems. We adapt the Guarded Commands language of Dijkstra [10] to more closely correspond to the primitives of the Timed Automata formalism – resulting in the *Timed Command* language – and we show how to obtain Timed Automata from programs in Timed Commands. We then develop a type system for enforcing an Information Flow policy on programs in Timed Commands – the main novelty being our treatment of non-determinism. We demonstrate the adequacy of the type system by means of a non-interference result [17,18]. Throughout we demonstrate the development on a simple voting protocol.

Related Work. There are other papers dealing with Information Flow on systems with a notion of time. Discrete time is considered in [11] that develops a non-interference property based on bisimulations of processes from a discrete time process algebra. A somewhat different direction in taken in [2] where a transformational type system is used to remove discrete timing as a covert channel for deterministic programs. Our contribution focuses on the challenges of continuous time and guarded actions of Timed Automata.

Continuous time is considered in [6] and [7] that present a notion of a timed non-interference for timed automata, while the work of [13] defines a notion of timed non-interference based on bisimulations for probabilistic timed automata. Our contribution considers a model closer to the Timed Automata of UPPAAL [16] and the development of a type system. A somewhat different approach is taken in [12] that studies the synthesis of controllers. Our key contribution is to develop a type system that prevents unnecessary *label creep* (where the boolean conditions passed exercise information flow to all variables subsequently used) and that deals with *non-determinism*, *non-termination* and *continuous real-time*.

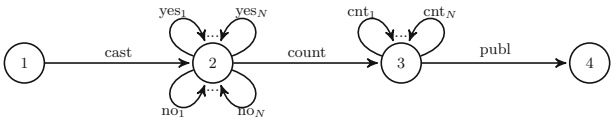
2 Timed Automata

A *Timed Automaton* [1,3] TA consists of a set of nodes Q, a set of annotated edges E, and a labelling function I on nodes. A node $q_\circ \in \mathsf{Q}$ will be the initial node and a node $q_\bullet \in \mathsf{Q}$ will be the final node; often $q_\bullet$ is intended not to be reachable. The mapping I maps each node in Q to a condition (to be introduced below) that will be imposed as an invariant at the node; we sometimes write $\mathsf{dom}(\mathsf{I})$ for Q and $\mathsf{TA} = (\mathsf{E}, \mathsf{I})$ or $\mathsf{TA} = (\mathsf{E}, \mathsf{I}, q_\circ, q_\bullet)$.

The edges are annotated with actions and take the form $(q_s, g \rightarrow act\colon \boldsymbol{r}, q_t)$ where act is given by

$$act ::= \boldsymbol{x} := \boldsymbol{e} \mid \texttt{publish}\ \boldsymbol{e}$$

and $q_s \in \mathsf{Q}$ is the source node and $q_t \in \mathsf{Q}$ is the target node. The action $g \rightarrow \boldsymbol{x} := \boldsymbol{e}\colon \boldsymbol{r}$ consists of a guard g that has to be satisfied in order for the multiple assignments $\boldsymbol{x} := \boldsymbol{e}$ to be performed and the clock variables $\boldsymbol{r}$ to be reset. We shall assume that the sequences $\boldsymbol{x}$ and $\boldsymbol{e}$ of program variables and



Invariants:

1	tt
2	t ≤ 50
3	t ≤ 30
4	tt

cast: $\rightarrow \mathtt{x}_1, ..., \mathtt{x}_N, \mathtt{y}_1, ..., \mathtt{y}_N, \mathtt{v}_1, ..., \mathtt{v}_N, \mathtt{c}{:=}\mathbf{0}{:}\ \mathtt{t}$
count: $\mathtt{t} = 50 \rightarrow :\ \mathtt{t}$
publ: $\mathtt{t} = 30 \rightarrow \mathtt{publish}\ \mathtt{c}{:}\ \mathtt{t}$

yes_i: $\mathtt{t} < 50 \wedge \mathtt{x}_i = 0 \rightarrow \mathtt{x}_i, \mathtt{v}_i{:=}1, 1{:}$
no_i: $\mathtt{t} < 50 \wedge \mathtt{x}_i = 0 \rightarrow \mathtt{x}_i, \mathtt{v}_i{:=}1, 0{:}$
cnt_i: $\mathtt{t} < 30 \wedge \mathtt{x}_i = 1 \wedge \mathtt{y}_i = 0 \rightarrow \mathtt{y}_i, \mathtt{c}{:=}1, \mathtt{c} + \mathtt{v}_i{:}$

Fig. 1. The timed automaton VP (and the abbreviations used).

expressions, respectively, have the same length and that $\boldsymbol{x}$ does not contain any repetitions. To cater for special cases we shall allow to omit the assignments of $g \rightarrow \boldsymbol{x} := \boldsymbol{e}{:}\ \boldsymbol{r}$ when $\boldsymbol{x}$ (and hence $\boldsymbol{e}$) is empty; also we shall allow to omit the guard g when it equals $\mathtt{tt}$ and to omit the clock resets when $\boldsymbol{r}$ is empty. The action $g \rightarrow \mathtt{publish}\ \boldsymbol{e}{:}\ r$ is fairly similar, the main difference being that no assignments are performed – the role of this action will become clear later when we discuss the security policies.

It has already emerged that we distinguish between (program) variables x and clock variables (or simply clocks) r. We write R for the set of clocks. The expressions e, guards g and conditions c are defined as follows using boolean tests b:

$$
\begin{aligned}
e &::= e_1\, \mathsf{op}_a\, e_2 \mid x \mid n \\
b &::= \mathtt{tt} \mid \mathtt{ff} \mid e_1\, \mathsf{op}_r\, e_2 \mid \neg b \mid b_1 \wedge b_2 \\
g &::= b \mid r\, \mathsf{op}_c\, n \mid (r_1 - r_2)\, \mathsf{op}_c\, n \mid g_1 \wedge g_2 \\
c &::= b \mid r\, \mathsf{op}_d\, n \mid (r_1 - r_2)\, \mathsf{op}_d\, n \mid c_1 \wedge c_2
\end{aligned}
$$

The arithmetic operators op_a and the relational operators op_r are as usual. For comparisons of clocks we use the operators $\mathsf{op}_c \in \{<, \leq, =\geq, >\}$ in guards and the less permissive set of operators $\mathsf{op}_d \in \{<, \leq, =\}$ in conditions.

Example 1. To illustrate our development we shall consider the voting protocol given by the timed automaton VP of Fig. 1. The protocol has N voters and three phases: casting (all edges leading to node number 2), counting (all edges leading to node number 3) and publishing (the edge leading to node number 4). For the casting phase a voter can choose to vote either yes ($\mathtt{v}_i = 1$) or no ($\mathtt{v}_i = 0$), or not to vote at all; $\mathtt{x}_i$ indicates whether or not the voter has voted. In the counting phase the votes are being counted using the variable $\mathtt{c}$; here $\mathtt{y}_i$ indicates whether the vote has been counted or not. Finally at the end of the counting phase the result is published. The clock $\mathtt{t}$ bounds the duration of the different phases of the protocol as expressed by the invariants of the nodes.

To specify the semantics of timed automata let σ be a state mapping variables to values (which we take to be integers) and let δ be a clock assignment mapping clocks to non-negative reals. We then have total semantic functions $[\![\cdot]\!]$ for evaluating the expressions, boolean tests, guards and conditions; the values

of the expressions and boolean expressions only depend on the states whereas that of guards and conditions also depend on the clock assignments.
The configurations of the timed automata have the form $\langle q, \sigma, \delta\rangle$ and we have transitions of two forms. Whenever $(q_s, g \to act\colon \boldsymbol{r}, q_t)$ is in E we have the *instant* rule:

$$\langle q_s, \sigma, \delta\rangle \longrightarrow \langle q_t, \sigma', \delta'\rangle \text{ if } \begin{cases} [\![g]\!](\sigma, \delta) = \mathsf{tt}, \\ \sigma' = [\![act]\!]\sigma, \delta' = \delta[\boldsymbol{r} \mapsto \mathbf{0}], \\ [\![\mathsf{I}(q_t)]\!](\sigma', \delta') = \mathsf{tt} \end{cases}$$

Whenever q is in Q we have a *delay* rule:

$$\langle q, \sigma, \delta\rangle \longrightarrow \langle q, \sigma, \delta'\rangle \text{ if } \begin{cases} \exists\, d > 0 : \delta' = \lambda r.\, \delta(r) + d, \\ [\![\mathsf{I}(q_s)]\!](\sigma, \delta') = \mathsf{tt} \end{cases}$$

The instant rule ensures that the guard is satisfied in the starting configuration and updates the mappings σ and δ and finally it ensures that the invariant is satisfied in the resulting configuration. Here the semantics of actions is given by $[\![\boldsymbol{x} := \boldsymbol{e}]\!]\sigma = \sigma[\boldsymbol{x} \mapsto [\![\boldsymbol{e}]\!]\sigma]$ (using the notation $\cdot[\cdot \mapsto \cdot]$) whereas $[\![\texttt{publish}\ \boldsymbol{e}]\!]\sigma = \sigma$. The delay rule only modifies the clock assignment with a delay d while ensuring that the invariant is satisfied in the resulting configuration. Initial configurations assume that all clocks are initialised to 0 and have the form $\langle q_\circ, \sigma, \lambda r.0\rangle$ where $[\![\mathsf{I}(q_\circ)]\!](\sigma, \lambda r.0) = \mathsf{tt}$.

Trace Behaviour. We do not want to admit Zeno behaviours nor do we want to admit systems that delay forever. We therefore combine the instant and delay rules into a *joint* rule that effectively first performs a number of delay rules (possibly none) and then an instant rule. So whenever $(q_s,\ g \to act\colon \boldsymbol{r},\ q_t)$ is in E we have:

$$\langle q_s, \sigma, \delta\rangle \implies \langle q_t, \sigma', \delta'\rangle \text{ if } \exists\, d \geq 0 : \begin{cases} [\![g]\!](\sigma, (\delta + d)) = \mathsf{tt}, \\ \sigma' = [\![act]\!]\sigma, \delta' = (\delta + d)[\boldsymbol{r} \mapsto \mathbf{0}], \\ [\![\mathsf{I}(q_s)]\!](\sigma, \delta + d) = \mathsf{tt}, [\![\mathsf{I}(q_t)]\!](\sigma', \delta') = \mathsf{tt} \end{cases}$$

where $\delta + d$ abbreviates $\lambda r.\, \delta(r) + d$. Here we use that it suffices to test the condition at the beginning and at the end of the periods of delay, because a condition c satisfies that if $[\![c]\!](\sigma, \delta)$ and $[\![c]\!](\sigma, \delta + d + d')$ for $d, d' \geq 0$ then also $[\![c]\!](\sigma, \delta + d)$.
We define a *trace* from $\langle q_s, \sigma, \delta\rangle$ to q_t in a timed automaton TA to have one of three forms. It may be a finite "successful" sequence

$$\begin{array}{l} \langle q_s, \sigma, \delta\rangle = \langle q'_0, \sigma'_0, \delta'_0\rangle \implies \cdots \implies \langle q'_n, \sigma'_n, \delta'_n\rangle \\ \text{such that } \{n\} = \{i \mid q'_i = q_t \wedge 0 < i \leq n\}. \end{array}$$

in which case at least one step is performed. It may be a finite "unsuccessful" sequence

$$\begin{array}{l} \langle q_s, \sigma, \delta\rangle = \langle q'_0, \sigma'_0, \delta'_0\rangle \implies \cdots \implies \langle q'_n, \sigma'_n, \delta'_n\rangle \\ \text{such that } \langle q'_n, \sigma'_n, \delta'_n\rangle \text{ is stuck and } q_t \notin \{q'_1, \cdots, q'_n\} \end{array}$$

where $\langle q'_n, \sigma'_n, \delta'_n \rangle$ is stuck when there is no joint action starting from $\langle q'_n, \sigma'_n, \delta'_n \rangle$. Finally, it may be an infinite "unsuccessful" sequence

$$\langle q_s, \sigma, \delta \rangle = \langle q'_0, \sigma'_0, \delta'_0 \rangle \Longrightarrow \cdots \Longrightarrow \langle q'_n, \sigma'_n, \delta'_n \rangle \Longrightarrow \cdots$$
$$\text{such that } q_t \notin \{q'_1, \cdots, q'_n, \cdots\}.$$

We may summarise the *trace behaviour* $[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta)$ of all traces from $\langle q_s, \sigma, \delta \rangle$ to q_t in the timed automaton TA by defining:

$$\begin{aligned}&[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta) = \\ &\quad \{(\sigma', \delta') \mid \text{a successful trace from } \langle q_s, \sigma, \delta \rangle \text{ to } q_t \text{ in } \mathsf{TA} \text{ ends in } \langle q_t, \sigma', \delta' \rangle\} \\ &\quad \cup \{\bot \mid \text{there is an unsuccessful trace from } \langle q_s, \sigma, \delta \rangle \text{ to } q_t \text{ in } \mathsf{TA}\}\end{aligned}$$

The only behaviour not accounted for by this definition is the potential delay in q_t and the potential joint actions starting from q_t.

3 Information Flow

We envisage that there is a security lattice expressing the permissible flows [9]. Formally this is a complete lattice and the permitted flows go in the direction of the partial order. In our development it will contain just two elements, L (for low) and H (for high), and we set $L \sqsubseteq H$ so that only the flow from H to L is disallowed. For confidentiality one would take L to mean public and H to mean private and for integrity one would take L to mean trusted and H to mean dubious. A more general development might consider a richer security lattice encompassing the Decentralized Label Model [14].

Example 2. Returning to the voting protocol of Example 1 we shall assume that the variables $\mathtt{x}_i$ (indicating whether or not the i'th participant has voted) and $\mathtt{y}_i$ (indicating whether or not the vote of the i'th participant has been counted) are public whereas the variables $\mathtt{v}_i$ (the actual vote of the i'the participant) and $\mathtt{c}$ (the result of the voting) are private. We shall consider it natural to let the clock $\mathtt{t}$ be public as well.

A *security policy* is then expressed by a mapping $\mathcal{L}$ that assigns an element of the security lattice to each program variable, clock variable, and node (i.e. program point). An entity is called *high* if it is mapped to H by $\mathcal{L}$, and it is said to be *low* if it is mapped to L by $\mathcal{L}$.

Example 3. Returning to the voting protocol of Examples 1 and 2 we shall let the security policy $\mathcal{L}$ map the variables $\mathtt{x}_i$ and $\mathtt{y}_i$ and the clock $\mathtt{t}$ to the low security level (L), while it maps $\mathtt{v}_i$ and $\mathtt{c}$ to the high security level (H). Furthermore, $\mathcal{L}$ maps all nodes to the low security level (L).

To express adherence to the security policy we use the binary operation $\rightsquigarrow$ defined on sets χ and χ' (of variables, clocks and nodes):

$$\chi \rightsquigarrow \chi' \Leftrightarrow \forall u \in \chi : \forall u' \in \chi' : \mathcal{L}(u) \sqsubseteq \mathcal{L}(u')$$

This expresses that all the entities of χ may flow into those of χ'; note that if one of the entities of χ has a high security level then it must be the case that all the entities of χ' have high security level.

Information flow control enforces a security policy by imposing constraints of the form $\{y\} \rightsquigarrow \{x\}$ whenever the value of y may somehow influence (or flow into) that of x. Traditionally we distinguish between *explicit* and *implicit* flows as explained below.

As an example of an *explicit* flow consider a simple assignment of the form $x{:=}e$. This gives rise to a condition $\mathsf{fv}(e) \rightsquigarrow \{x\}$ so as to indicate that the *explicit* flow from the variables of e to the variable x must adhere to the security policy: if e contains a variable with high security level then x also must have high security level.

For an example of an *implicit* flow consider a conditional assignment $g \rightarrow x{:=}0$ where x is assigned the constant value 0 in case g evaluates to true. This gives rise to a condition $\mathsf{fv}(g) \rightsquigarrow \{x\}$ so as to indicate that the *implicit* flow from the variables of g to the variable x must adhere to the security policy: if g contains a variable with high security level then x also must have high security level. (If used indiscriminately this gives rise to *label creep* where variables tend to have to be given the high security classification.)

In this paper we develop an approach to ensuring that the security policy is adhered to by the Timed Automaton of interest. The key idea is to ensure that $\{x\} \rightsquigarrow \{y\}$ whenever there is an *explicit* flow of information from x to y (as illustrated above) or an *implicit* flow from x to y; traditionally, implicit flows arise because of testing guards and conditions, but we shall see that the highly nondeterministic nature of Timed Automata provide yet another contribution. We shall say that we prevent information flows from high variables to low variables.

To overcome the vagueness of this explanation we need to define a semantic condition that encompasses our notion of permissible information flow. We begin by defining $(\sigma, \delta) \equiv (\sigma', \delta')$ to indicate that the two pairs are equal on low variables and low clocks:

$$(\sigma,\delta) \equiv (\sigma',\delta') \quad \text{iff} \quad \begin{array}{l} \forall x : \mathcal{L}(x) = L \Rightarrow \sigma(x) = \sigma'(x) \wedge \\ \forall r : \mathcal{L}(r) = L \Rightarrow \delta(r) = \delta'(r) \end{array}$$

To cater for the $\perp$ behaviour produced by the trace behaviour we shall allow to write $\perp \equiv \perp$ and take it for granted that $\perp \not\equiv (\sigma, \delta)$ and $(\sigma, \delta) \not\equiv \perp$. It is immediate that this definition of $\equiv$ gives rise to an equivalence relation.

We next lift the operation $\equiv$ to work on sets:

$$\Gamma \equiv \Gamma' \quad \text{iff} \quad \begin{array}{l} \forall \gamma \in \Gamma : \exists \gamma' \in \Gamma' : \gamma \equiv \gamma' \wedge \\ \forall \gamma' \in \Gamma' : \exists \gamma \in \Gamma : \gamma \equiv \gamma' \end{array}$$

Here γ ranges over pairs (σ, δ) as well as $\perp$, and it is immediate that this definition of $\equiv$ gives rise to an equivalence relation.

We can now express our semantic condition for when a Timed Automaton $\mathsf{TA} = (\mathsf{E}, \mathsf{I})$ satisfies the Information Flow security policy by the condition:

$$\begin{array}{l}(\sigma,\delta) \equiv (\sigma',\delta') \ \wedge\ [\![\mathsf{I}(q_\circ)]\!](\sigma,\delta) \ \wedge\ [\![\mathsf{I}(q_\circ)]\!](\sigma',\delta') \\ \Downarrow \\ [\![(\mathsf{E},\mathsf{I}) : q_\circ \mapsto q_\bullet]\!](\sigma,\delta) \equiv [\![(\mathsf{E},\mathsf{I}) : q_\circ \mapsto q_\bullet]\!](\sigma',\delta')\end{array}$$

It says that if we consider two initial configurations that only differ on high variables and clocks then the final configurations are also only allowed to differ on high variabels and clocks; it is immediate that the final configurations (except $\perp$) also satisfy $\mathsf{I}(q_\bullet)$. In other words, there is no information flow from the initial values of high variables and clocks to the final values of low variables and clocks. The fact that the trace behaviour produces a set of configurations means that we take due care of non-determinism, and the fact that the trace behaviour may contain $\perp$ means that we take due care of non-termination (be it because of looping or because of getting stuck).

This semantic condition is more involved than in classical papers like [17] due to the highly non-deterministic nature of Timed Automata. As an example of the difficulties of treating non-determinism, the previous attempt of [5] is flawed because a command may terminate as well as loop – this was pointed out in [17, Sect. 7] which therefore performs a development for deterministic programs only. For another example, illustrating one of the problems solved by our type system, consider the program $y > 0 \rightarrow \mathit{skip}\ []\ \mathtt{tt} \rightarrow x := 0$ making a non-deterministic choice between two guarded actions. Writing $x \not\rightsquigarrow y$ to indicate that y does not depend on x, the type system of [5] allows to establish

$$\vdash_1 \{x \not\rightsquigarrow y, y \not\rightsquigarrow x\}\ y > 0 \rightarrow \mathit{skip}\ []\ \mathtt{tt} \rightarrow x := 0\ \{y \not\rightsquigarrow x\}$$

which is unsound. To see this note that for $\sigma_1 = [y \mapsto 1, x \mapsto 2]$ the final values of x can be 0 and 2, while for $\sigma_2 = [y \mapsto 0, x \mapsto 2]$, the final value of x can only be 0.

4 Timed Commands

The semantic condition for Information Flow is undecidable in general. To obtain a sound and decidable enforcement mechanism, the traditional approach is to develop a type system for a suitable programming language or process calculus. To this end we introduce the language TC of *Timed Commands*. It is strongly motivated by Dijkstra's language of Guarded Commands [10] but is designed so that it combines guards and assignments in the manner of Timed Automata. The syntax is given by:

$$\begin{array}{rcl} TC & ::= & \mathtt{begin}^{[c_\circ]}\ C\ {}^{[c_\bullet]}\mathtt{end} \\ C & ::= & g \rightarrow \mathit{act}\colon \boldsymbol{r} \ \mid\ C_1 ;^{[c]} C_2 \ \mid\ \mathtt{do}\ T_1\ []\ \cdots\ []\ T_n\ \mathtt{od}\ []\ T_{n+1}\ []\ \cdots\ []\ T_m \\ T & ::= & g \rightarrow \mathit{act}\colon \boldsymbol{r} \ \mid\ T ;^{[c]} C \end{array}$$

A timed command TC specifies a condition $c_\circ$ that must hold initially and a condition $c_\bullet$ that must hold if the command terminates. The command C itself

can have one of three forms. One possibility is that it is an action of the form $g \to act\colon \boldsymbol{r}$. Another possibility is that it is a sequence of commands and then the condition c must be satisfied when moving from the first command to the second. The third possibility is that it is a looping construct with a number of branches $T_1, \cdots, T_n$ that will loop and a number of branches $T_{n+1}, \cdots, T_m$ that will terminate the looping behaviour. In case $n = 0$ and $m > 1$ we allow to dispense with the `do od`. Here T is a special form of command that starts with an action and potentially is followed by a number of commands. Conditions, guards and expressions are defined as in Sect. 2.

Example 4. Using the abbreviations of Fig. 1 the voting protocol of Example 1 is given by the following timed command:

$$\begin{array}{l}\texttt{begin}^{[\mathrm{tt}]}\ \mathrm{cast};^{[\mathrm{t}\le 50]}\\ \qquad (\texttt{do}\ \mathrm{yes}_1\ [\!]\ \cdots\ [\!]\ \mathrm{yes}_N\ [\!]\ \mathrm{no}_1\ [\!]\ \cdots\ [\!]\ \mathrm{no}_N\ \texttt{od}\ [\!]\ \mathrm{count})\ ;^{[\mathrm{t}\le 30]}\\ \qquad (\texttt{do}\ \mathrm{cnt}_1\ [\!]\ \cdots\ [\!]\ \mathrm{cnt}_N\ \texttt{od}\ [\!]\ \mathrm{publ})\\ {}^{[\mathrm{tt}]}\texttt{end}\end{array}$$

The first line performs the initialisation for the casting phase which happens in the second line; the third line expresses the counting of the votes and their publication. The timing constraints are expressed in the superscripts.

Transformational Semantics. We shall define the semantics of a timed command by mapping it into a timed automaton. Consider $\texttt{begin}^{[c_\circ]}\ C\ {}^{[c_\bullet]}\texttt{end}$ and let $q_\circ$ and $q_\bullet$ be two disctinct nodes; they will be the initial and final node of the resulting timed automaton and we shall ensure that $\mathsf{l}(q_\circ) = c_\circ$ and $\mathsf{l}(q_\bullet) = c_\bullet$. Additional nodes will be created during the construction using a judgement of the form:

$$\vdash_{q_s}^{q_t} C : \mathsf{E}, \mathsf{l}$$

$$\vdash_{q_s}^{q_t} g \to act\colon \boldsymbol{r} : \{(q_s, g \to act\colon \boldsymbol{r}, q_t)\}, [\,]$$

$$\frac{\vdash_{q_s}^{q} C_1 : \mathsf{E}_1, \mathsf{l}_1 \qquad \vdash_{q}^{q_t} C_2 : \mathsf{E}_2, \mathsf{l}_2}{\vdash_{q_s}^{q_t} C_1;^{[c]} C_2 : \mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{l}_1 \cup \mathsf{l}_2 \cup [q \mapsto c]} \quad \text{where } q \text{ is fresh}$$

$$\frac{\bigwedge_{i=1}^{n} \vdash_{q_s}^{q_s} T_i : \mathsf{E}_i, \mathsf{l}_i \qquad \bigwedge_{i=n+1}^{m} \vdash_{q_s}^{q_t} T_i : \mathsf{E}_i, \mathsf{l}_i}{\vdash_{q_s}^{q_t} \texttt{do}\ T_1\ [\!]\ \cdots\ [\!]\ T_n\ \texttt{od}\ [\!]\ T_{n+1}\ [\!]\ \cdots\ [\!]\ T_m : \bigcup_i \mathsf{E}_i, \bigcup_i \mathsf{l}_i}$$

$$\frac{\vdash_{q_\circ}^{q_\bullet} C : \mathsf{E}, \mathsf{l}}{\vdash \texttt{begin}^{[c_\circ]}\ C\ {}^{[c_\bullet]}\texttt{end} : \mathsf{E}, \mathsf{l}', q_\circ, q_\bullet} \quad \text{where } \begin{cases} \mathsf{l}' = \mathsf{l}[q_\circ \mapsto c_\circ; q_\bullet \mapsto c_\bullet] \\ q_\circ,\ q_\bullet \text{ are fresh} \end{cases}$$

Fig. 2. From timed commands to timed automata.

Here C is a timed command, q_s and q_t are nodes, E is a set of edges, and the judgement will introduce additional nodes whose invariants are given by the labelling function I. This defines a timed automaton with initial node q_s, final node q_t, edges E, and labelling function I.

The judgement is specified by the axioms and rules of Fig. 2. In the axiom we simply create the edge $(q_s, g \rightarrow act\colon \boldsymbol{r}, q_t)$ starting in q_s and ending in q_t and indicating the action to be performed; the resulting labelling function is empty as no new nodes are created in the construct.

In the first rule we create a *fresh* node q to be used to glue the timed automata for C_1 and C_2 together; the node q has the invariant c and is used as target node for C_1 as well as source node for C_2. The resulting set of edges is the union of the two sets; the two branches will create disjoint sets of nodes so the two mappings I_1 and I_2 will have disjoint domains and we write union for their combination.

In the rule for the looping construct we achieve the looping of the branches $T_1, \cdots, T_n$ by using q_s as source as well as target node, whereas for $T_{n+1}, \cdots, T_m$ we use q_t as target node. The overall set of edges are obtained as the union of the edges E_i and as in the previous case the domains of the mappings I_i will be disjoint so the mappings are easily combined.

Recall that T is a special form of timed command and hence timed automata can be constructed using the judgements of Fig. 2. The timed automata constructed from T always have exactly one edge leaving the initial node and do not contain any edge back to the initial node unless the initial and final nodes coincide. The timed automata constructed from C may have more than one edge leaving the initial node and may contain edges back to the initial node even when the initial and final nodes are distinct.

For the overall timed command $\texttt{begin}^{[c_\circ]}\ C\ ^{[c_\bullet]}\texttt{end}$ we can now obtain a timed automaton with initial node $q_\circ$, final node $q_\bullet$, and edges E, and labelling function I' given by the last inference rule of Fig. 2.

Example 5. The transformation applied to the timed command of Example 4 gives rise to the timed automata of Fig. 1.

5 Type System

The information flow type system is specified using judgements of the form

$$\vdash^{[q_t:c_t]}_{[q_s:c_s]} C : \mathsf{E}, \mathsf{I} \,\&\, \chi$$

This is an extension of the judgements $\vdash^{q_t}_{q_s} C : \mathsf{E}, \mathsf{I}$ of the previous section for constructing timed automata from commands. The new judgements maintain information about the invariants c_s and c_t associated with the nodes q_s and q_t and a set χ of *latent variables and nodes* that influence the termination of the command; the influence of χ on q_t remains to be enforced. The type system is specified in Fig. 3 and explained below.

Assignment. Consider the first axiom of Fig. 3. The second line of the side condition expresses all the explicit flows from components of the vector of expressions

$$\vdash^{[q_t:c_t]}_{[q_s:c_s]} g \to \boldsymbol{x} := \boldsymbol{e}\colon \boldsymbol{r} : \{(q_s, g \to \boldsymbol{x} := \boldsymbol{e}\colon \boldsymbol{r}, q_t)\}, [\,] \,\&\, \{q_s\} \cup \mathsf{fv}(c_s \wedge g \wedge c_t[\boldsymbol{e}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}])$$

if $\{q_s\} \rightsquigarrow \{q_t, \boldsymbol{x}, \boldsymbol{r}\}$
$\bigwedge_i \mathsf{fv}(e_i) \rightsquigarrow \{x_i\}$
$\mathsf{fv}(c_s \wedge g \wedge c_t[\boldsymbol{e}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]) \rightsquigarrow \{\boldsymbol{x}, \boldsymbol{r}\}$

$$\vdash^{[q_t:c_t]}_{[q_s:c_s]} g \to \texttt{publish}\ \boldsymbol{e}\colon \boldsymbol{r} : \{(q_s, g \to \texttt{publish}\ \boldsymbol{e}\colon \boldsymbol{r}, q_t)\}, [\,] \,\&\, \{q_s\} \cup \mathsf{fv}(c_s \wedge g \wedge c_t[\boldsymbol{0}/\boldsymbol{r}])$$

if $\{q_s\} \rightsquigarrow \{q_t, \boldsymbol{r}\}$
$\mathsf{fv}(c_s \wedge g \wedge c_t[\boldsymbol{0}/\boldsymbol{r}]) \rightsquigarrow \{\boldsymbol{r}\}$

$$\frac{\vdash^{[q:c]}_{[q_s:c_s]} C_1 : \mathsf{E}_1, \mathsf{I}_1 \,\&\, \chi_1 \qquad \vdash^{[q_t:c_t]}_{[q:c]} C_2 : \mathsf{E}_2, \mathsf{I}_2 \,\&\, \chi_2}{\vdash^{[q_t:c_t]}_{[q_s:c_s]} C_1 ;^{[c]} C_2 : \mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{I}_1 \cup \mathsf{I}_2 \cup [q \mapsto c] \,\&\, \chi_2}$$

if q is fresh
$\mathsf{fv}(c) \cup \{q\} \rightsquigarrow R \cup \{q\}$
$\chi_1 \rightsquigarrow \{q\}$

$$\frac{\bigwedge^n_{i=1} \vdash^{[q_s:c_s]}_{[q_s:c_s]} T_i : \mathsf{E}_i, \mathsf{I}_i \,\&\, \chi_i \qquad \bigwedge^m_{i=n+1} \vdash^{[q_t:c_t]}_{[q_s:c_s]} T_i : \mathsf{E}_i, \mathsf{I}_i \,\&\, \chi_i}{\vdash^{[q_t:c_t]}_{[q_s:c_s]} \texttt{do}\ T_1\ [\!]\ \cdots\ [\!]\ T_n\ \texttt{od}\ [\!]\ T_{n+1}\ [\!]\ \cdots\ [\!]\ T_m : \bigcup_i \mathsf{E}_i, \bigcup_i \mathsf{I}_i \,\&\, \{q_t\}}$$

if $\{q_s\} \rightsquigarrow \{q_t\}$
$\bigwedge^n_{i=1} \chi_i \rightsquigarrow \{q_s\}$
$\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}] \Rightarrow \bigwedge^m_{i=n+1} \chi_i \rightsquigarrow \{q_t\}$
$\bigwedge_{i,j \mid i \neq j, \underline{\mathsf{sat}}(\mathsf{fst}^{\zeta_i}_{c_s}(T_i) \wedge \mathsf{fst}^{\zeta_j}_{c_s}(T_j))} \chi_i \rightsquigarrow \mathsf{ass}(T_j)$
where ζ_l is c_s if $l \leq n$ and ζ_l is c_t if $l > n$
$\bigwedge^m_{i=n+1} (\forall r \in \mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T_i)) \cap R : \mathcal{L}(r) = L \wedge \bigwedge^m_{j=n+1} \overline{\mathsf{fst}^{c_t}_{c_s}(T_i)} \Leftrightarrow \overline{\mathsf{fst}^{c_t}_{c_s}(T_j)})$

$$\frac{\vdash^{[q_\bullet:c_\bullet]}_{[q_\circ:c_\circ]} C : \mathsf{E}, \mathsf{I} \,\&\, \chi}{\vdash \texttt{begin}^{[c_\circ]}\ C\ {}^{[c_\bullet]}\texttt{end} : \mathsf{E}, \mathsf{I}', q_\circ, q_\bullet}$$

where
$\mathsf{I}' = \mathsf{I}[q_\circ \mapsto c_\circ; q_\bullet \mapsto c_\bullet]$
$\mathsf{fv}(c_\circ) \cup \{q_\circ\} \rightsquigarrow R \cup \{q_\circ\}$
$\mathsf{fv}(c_\bullet) \cup \{q_\bullet\} \rightsquigarrow R \cup \{q_\bullet\}$
$\chi \rightsquigarrow \{q_\bullet\}$
$\mathcal{L}(q_\bullet) = L$
$q_\circ$, $q_\bullet$ are fresh

Fig. 3. Type system for timed commands.

to corresponding components of the vector of variables. The first line of the side condition expresses that the modifications of variables and clocks as well as the termination relies on having started the action. The third line of the side condition expresses our knowledge that c_s holds and the implict flows arising from testing the guard g in the pre-state and the condition c_t in the post-state before performing the modifications of variables and clocks. (We are using the insight

from Hoare logic [4] that evaluating c_t in the post-state is the same as evaluating $c_t[\boldsymbol{e}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]$ in the pre-state.) Rather than also expressing the implicit flow for termination (in the form of a side condition $\mathsf{fv}(c_s \wedge g \wedge c_t[\boldsymbol{e}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]) \rightsquigarrow \{q_t\}$) we produce the latent set of variables and nodes $\{q_s\} \cup \mathsf{fv}(c_s \wedge g \wedge c_t[\boldsymbol{e}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}])$ as listed after the ampersand in the axiom. (We shall see the flexibiliity offered by this approach shortly.)

Example 6. Consider the action cnt_i of Fig. 1. It will be the case that $q_s = q_t = 3$ and $c_s = c_t = \mathtt{t} \leq 30$. The type system imposes the following constraints on the flows:

$$\{3\} \rightsquigarrow \{3, \mathtt{y}_i, \mathtt{c}\}, \quad \{\,\} \rightsquigarrow \{\mathtt{y}_i\}, \quad \{\mathtt{c}, \mathtt{v}_i\} \rightsquigarrow \{\mathtt{c}\}, \quad \{\mathtt{t}, \mathtt{x}_i, \mathtt{y}_i\} \rightsquigarrow \{\mathtt{y}_i, \mathtt{c}\}$$

It is easy to check that they are fulfilled for the security assignment of Example 3. The latent set of variables is $\{3, \mathtt{t}, \mathtt{x}_i, \mathtt{y}_i\}$.

Publish. The second axiom of Fig. 3 is a simplification of the first axiom in that the values computed are "published" but not recorded in the state. (The main purpose of this rule is to "bypass" the security policy in that we allow the publication of expressions even when they contain high variables.)

Example 7. For the action publ of Fig. 1 we have $q_s = 3$, $q_t = 4$, $c_s = \mathtt{t} \leq 30$ and $c_t = \mathtt{tt}$. The type system impose the contraints $\{3\} \rightsquigarrow \{4, \mathtt{t}\}$ and $\{\mathtt{t}\} \rightsquigarrow \{\mathtt{t}\}$ which clearly hold with the security assignment of Example 3.

Sequence. The first inference rule of Fig. 3 deals with the sequential composition of two commands. The second line of the side condition expresses the explicit flow possible due to the delay at the node q separating the two commands; here R is the set of all clock variables and it is included to mimick the effect of the potential delay. The third line of the side condition takes care of imposing the latent effect of the first command on the node q following immediately after it.

Example 8. Let us consider the sequencing construct $;^{[\mathtt{t} \leq 30]}$ between the two loops of the command of Example 4. The latent set of variables from the first loop will simply be $\{3\}$ and the two constraints will amount to $\{\mathtt{t}, 3\} \rightsquigarrow \{\mathtt{t}, 3\}$ and $\{3\} \rightsquigarrow \{3\}$ which are satisfied for the security assignment of Example 3.

Auxiliary Operations. Before approaching the last inference rule in Fig. 3 we shall introduce three auxiliary operations.

The auxiliary operation $\mathsf{ass}(C)$ overapproximates the set of variables and clocks modified by the command (ignoring any initial and final delays):

$$\begin{aligned}
\mathsf{ass}(g \to \boldsymbol{x} := \boldsymbol{e}\colon \boldsymbol{r}) &= \{\boldsymbol{x}, \boldsymbol{r}\} \\
\mathsf{ass}(g \to \mathtt{publish}\ \boldsymbol{e}\colon \boldsymbol{r}) &= \{\boldsymbol{r}\} \\
\mathsf{ass}(C_1 ;^{[c]} C_2) &= \mathsf{ass}(C_1) \cup \mathsf{ass}(C_2) \cup R \\
\mathsf{ass}\left(\begin{array}{c}\mathtt{do}\ T_1\ [\!]\ \cdots\ [\!]\ T_n\ \mathtt{od} \\ [\!]\ T_{n+1}\ [\!]\ \cdots\ [\!]\ T_m\end{array}\right) &= \begin{cases}\mathsf{ass}(T_1) \cup \cdots \cup \mathsf{ass}(T_m) \cup R & \text{if } n > 0 \\ \mathsf{ass}(T_1) \cup \cdots \cup \mathsf{ass}(T_m) & \text{if } n = 0\end{cases}
\end{aligned}$$

where R is the set of all clocks and it is included to mimick the effect of the potential (internal) delays of the loop.

Fact 1. *If* $\vdash^{q_t}_{q_s} C : \mathsf{E}, \mathsf{I}$ *and if* $(\sigma', \delta') \in [\![(\mathsf{E}, \mathsf{I}[q_s \mapsto c_s][q_t \mapsto c_t] : q_s \mapsto q_t]\!](\sigma, \delta)$ *then* $\exists d \geq 0 : \{x \mid \sigma(x) \neq \sigma'(x)\} \cup \{r \mid \delta(r) + d \neq \delta'(r)\} \subseteq \mathsf{ass}(C)$, *where* d *corresponds to the initial delay.*

The auxiliary operation $\mathsf{fst}^{c_t}_{c_s}(T)$ determines the initial guard and the condition immediately following it (in the manner of the rule for assignment):

$$\begin{aligned} \mathsf{fst}^{c_t}_{c_s}(g \rightarrow \boldsymbol{x} := \boldsymbol{e} \colon \boldsymbol{r}) &= c_s \wedge g \wedge c_t[\boldsymbol{e}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}] \\ \mathsf{fst}^{c_t}_{c_s}(g \rightarrow \mathtt{publish}\ \boldsymbol{e} \colon \boldsymbol{r}) &= c_s \wedge g \wedge c_t[\boldsymbol{0}/\boldsymbol{r}] \\ \mathsf{fst}^{c_t}_{c_s}(T ;^{[c]} C) &= \mathsf{fst}^{c}_{c_s}(T) \end{aligned}$$

The inclusion of c_s is so as to get the strongest information for use in the rule for the looping construct in Fig. 3.

We shall need the auxiliary predicate $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}]$ that must be true whenever it is possible that the construct $\mathtt{do}\ T_1\ [\!]\ \cdots\ [\!]\ T_n\ \mathtt{od}\ [\!]\ T_{n+1}\ [\!]\ \cdots\ [\!]\ T_m$ does *not* terminate from a state satisfying c_s; we return to this below.

Looping. We can now explain the inference rule in Fig. 3 for looping. The first line in the side condition expresses that the termination relies on having started the action as we saw in the axiom for assignment. The second line in the side condition takes care of imposing the latent effect χ_i of the looping commands on the loop header q_s.

The third line in the side condition takes care of imposing the latent effect of the terminating commands on the final node q_t. However, by using the predicate $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}]$ we allow to dispense with imposing this latent effect in case termination of the looping construct is guaranteed. As an example this means that the type system will allow the following Timed Command

$$\big(\,(\mathtt{h} = \mathtt{0} \rightarrow \mathtt{h}{:=}\mathtt{h}\colon\ )\ [\!]\ (\mathtt{h} \neq \mathtt{0} \rightarrow \mathtt{h}{:=}\mathtt{h}\colon\ )\,\big)\ ;^{[\mathtt{tt}]}\ \mathtt{tt} \rightarrow \mathtt{l}{:=}\mathtt{l}\colon$$

that would otherwise be disallowed (assuming that `h` is a high variable and `l` is a low variable). Indeed it is in order to accomodate this kind of behaviour that the type system makes use of latent variables and nodes. This is essential for preventing unnecessary *label creep* where programs operating on high data too often end up in a high control point.

Using the notation of Fig. 3 we can now clarify our demands on the auxiliary notation $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}]$ used in the third line:

$$\begin{array}{l} \bot \in \bigcup_{(\sigma,\delta)\mid [\![c_s]\!](\sigma,\delta)} [\![(\cup_i \mathsf{E}_i, \cup_i \mathsf{I}_i[q_s \mapsto c_s][q_t \mapsto c_t]) : q_s \mapsto q_t]\!](\sigma,\delta) \\ \Downarrow \\ \Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}] \end{array}$$

The subscript $(\sigma, \delta) \mid [\![c_s]\!](\sigma, \delta)$ is intended to let (σ, δ) range over all possibilities that satisfy $[\![c_s]\!](\sigma, \delta)$. Note that we do not require to capture non-termination precisely but will allow any over-approximation.

Before explaining the fourth line in the side condition it is helpful to establish the following property of the type system as stated in Fig. 3.

Lemma 1. *If* $\vdash^{[q_t:c_t]}_{[q_s:c_s]} C : \mathsf{E}, \mathsf{I} \,\&\, \chi$ *then we have that* $\{q_s\} \rightsquigarrow \mathsf{ass}(C) \cup \{q_t\}$ *and* $\forall \chi' : (\chi \rightsquigarrow \chi') \Rightarrow (\{q_s\} \rightsquigarrow \chi')$.

If $\vdash^{[q_t:c_t]}_{[q_s:c_s]} T : \mathsf{E}, \mathsf{I} \,\&\, \chi$ *then* $\forall \chi' : (\chi \rightsquigarrow \chi') \Rightarrow (\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T)) \rightsquigarrow \chi')$ *and* $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T)) \rightsquigarrow \mathsf{ass}(T)$, *and* $\{q_s\} \rightsquigarrow \{q_t\}$.
(Note that the lack of reflexivity of $\rightsquigarrow$ means that we need to write slightly complex formulae like $\forall \chi' : (\chi \rightsquigarrow \chi') \Rightarrow ((\cdots) \rightsquigarrow \chi')$ because the formula $((\cdots) \rightsquigarrow \chi$ is in general incorrect.)

Proof. We prove the first statement by induction on $\vdash^{[q_t:c_t]}_{[q_s:c_s]} C : \mathsf{E}, \mathsf{I} \,\&\, \chi$ using that $\rightsquigarrow$ is transitive.

We prove the second statement by induction on $\vdash^{[q_t:c_t]}_{[q_s:c_s]} T : \mathsf{E}, \mathsf{I} \,\&\, \chi$. It is immediate for the two axioms for actions because $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T)) = \chi$. In the rule for composition for $T;^{[c]}C$ observe that $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T;^{[c]}C)) = \{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c}_{c_s}(T))$ and that the induction hypothesis gives that $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c}_{c_s}(T)) \rightsquigarrow \{q\}$ because $\chi_1 \rightsquigarrow \{q\}$. We have $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c}_{c_s}(T)) \rightsquigarrow \mathsf{ass}(T)$ from the induction hypothesis, $\{q\} \rightsquigarrow R$ from the rule, and $\{q\} \rightsquigarrow \mathsf{ass}(C)$ from the previous result, and then get $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T;^{[c]}C)) \rightsquigarrow \mathsf{ass}(T;^{[c]}C)$. Next suppose $\chi = \chi_2 \rightsquigarrow \chi'$; from the previous result we have $\{q\} \rightsquigarrow \chi'$ and hence $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T;^{[c]}C)) \rightsquigarrow \chi'$.

This lemma shows that we have already taken care of the so-called *block labels* of [9] and thereby take care of the implicit flows due to testing guards and conditions in the manner of [17]. However, the language considered in [17] is deterministic and the presence of non-determinism in Timed Commands poses a complication as illustrated by the following command:

$$\mathtt{tt} \rightarrow \mathtt{l{:=}0{:}} \;;^{[\mathtt{tt}]} \big((\mathtt{h} = 0 \rightarrow \mathtt{h{:=}h{:}}) \;[]\; (\mathtt{tt} \rightarrow \mathtt{l{:=}1{:}}) \big)$$

Here the final value of `l` will be 1 if $\mathtt{h} \neq 0$, but the final value of `l` may be either 0 or 1 if $\mathtt{h} = 0$. This presents a violation of our semantic conditions for adherence to the Information Flow security policy.

The purpose of the fourth line in the side condition is to take care of this possibility and this is a novel contribution with respect to [5,9,17] as discussed in Sect. 3. The notation $\underline{\mathsf{sat}}(\cdots)$ is intended to express the satisfiability of the $\cdots$ formula. We are considering all terminating branches in the looping construct and whenever there are two branches that are not mutually exclusive (that is, where $\underline{\mathsf{sat}}(\mathsf{fst}^{\zeta_i}_{c_s}(T_i) \wedge \mathsf{fst}^{\zeta_j}_{c_s}(T_j)))$ we make sure to record the information flow arising from *bypassing* the branch that would otherwise perform an assignment. This is essential for dealing with *non-determinism* and *non-termination*.

Before explaining the fifth condition let us consider the following command operating on a low clock `l` and a high clock `h`:

$$\mathtt{tt} \rightarrow : l;^{[\mathtt{tt}]} \big(\mathtt{do\ od}\ [] \ \mathtt{h} \geq 100 \rightarrow : \big)$$

Here we have that $(\sigma, \delta[\mathtt{h} \mapsto 110]) \equiv (\sigma, \delta[\mathtt{h} \mapsto 90])$ but running the command from $(\sigma, \delta[\mathtt{h} \mapsto 110])$ might produce $(\sigma, \delta[\mathtt{h} \mapsto 110])$ ifself whereas running the command from $(\sigma, \delta[\mathtt{h} \mapsto 90])$ can only produce $(\sigma, (\delta[\mathtt{h} \mapsto 90]) + d)$ for $d \geq 10$ in which case $(\sigma, \delta[\mathtt{h} \mapsto 110]) \not\equiv (\sigma, (\delta[\mathtt{h} \mapsto 90]) + d)$.

The purpose of the fifth line in the side condition is to take care of this possibility by enforcing that the terminating branches only test on low clocks and that the conditions on clocks are the same. To this end we define $\overline{g}$ as follows

$$\begin{aligned} \overline{b} &= \mathsf{tt} \\ \overline{r\ \mathsf{op}_c\ n} &= r\ \mathsf{op}_c\ n \\ \overline{(r_1 - r_2)\ \mathsf{op}_c\ n} &= (r_1 - r_2)\ \mathsf{op}_c\ n \\ \overline{g_1 \wedge g_2} &= \overline{g_1} \wedge \overline{g_2} \end{aligned}$$

and we write $g \Leftrightarrow g'$ to express the equivalence of the guards g and g'. This is essential for the type system to deal correctly with the *continuous clocks*.

Example 9. Returning to Example 4 let us consider the looping command of the third line. Using the latent set of variables from Example 6 we obtain the following constraints from the first two lines of the condition:

$$\{3\} \rightsquigarrow \{4\}, \quad \{3, \mathtt{t}, \mathtt{x}_1, \ldots, \mathtt{x}_N, \mathtt{y}_1, \ldots, \mathtt{y}_N\} \rightsquigarrow \{3\}$$

We have no contribution from the third side condition of the rule since termination of the loop is guaranteed. From the fourth side condition we get

$$\bigcup_{i \neq j} \{3, \mathtt{t}, \mathtt{x}_i, \mathtt{y}_i\} \rightsquigarrow \{\mathtt{y}_j, \mathtt{c}\}$$

and from the fifth line we get $\mathcal{L}(\mathtt{t}) = L$. It is easy to check that the above conditions are fulfilled with the security assignment of Example 3.

Timed Commands. Consider the last inference rule in Fig. 3. The first and last lines of the side condition are as in Fig. 2. The second and third lines of the side condition express the explicit flow possible due to the delay at the node $q_\circ$ and $q_\bullet$ and is analogous to our treatment of sequencing. The fourth line of the side condition takes care of imposing the latent effect of the command on the final node q_t and is analogous to our treatment of sequencing. The fifth line will allow us to invoke Theorem 1 of the next section.

6 Adequacy

To prove the adequacy of the type system we shall establish some terminology. A function like $[\![\mathsf{TA} : q_s \mapsto q_t]\!]$ mapping a pair of state and clock assignment to a set of pairs of states and clock assignments and possibly the symbol $\perp$ will be called a *semantic function*. Whenever F is a semantic function we define

$$F \models c_s \mapsto c_t \text{ iff } \forall (\sigma, \delta), (\sigma', \delta') : \begin{array}{l} (\sigma, \delta) \equiv_{c_s} (\sigma', \delta') \\ \Downarrow \\ F(\sigma, \delta) \equiv^{c_t} F(\sigma', \delta') \end{array}$$

where (using $\equiv$ as defined in Sect. 3)

$$(\sigma,\delta) \equiv_c (\sigma',\delta') \text{ abbreviates } (\sigma,\delta) \equiv (\sigma',\delta') \ \wedge\ [\![c]\!](\sigma,\delta) \ \wedge\ [\![c]\!](\sigma',\delta')$$
$$\Gamma \equiv^c \Gamma' \text{ abbreviates } \Gamma \equiv \Gamma' \wedge \forall(\sigma,\delta) \in \Gamma : [\![c]\!](\sigma,\delta) \ \wedge\ \forall(\sigma',\delta') \in \Gamma' : [\![c]\!](\sigma',\delta')$$

The semantic condition for when a Timed Automaton $\mathsf{TA} = (\mathsf{E}, \mathsf{I}, q_\circ, q_\bullet)$ satisfies the Information Flow security policy discussed in Sect. 3 then amounts to $[\![(\mathsf{E},\mathsf{I}) : q_\circ \mapsto q_\bullet]\!] \models \mathsf{I}(q_\circ) \mapsto \mathsf{I}(q_\bullet)$. Finally, let us define the composition of two semantic functions F_1 and F_2 as follows:

$$F_1 \diamond F_2 = \lambda(\sigma_0,\delta_0).\ (F_1(\sigma_0,\delta_0) \cap \{\bot\}) \cup \bigcup_{(\sigma_1,\delta_1)\in F(\sigma_0,\delta_0)\setminus\{\bot\}} F_2(\sigma_1,\delta_1)$$

Fact 2. *If* $F_1 \models c_0 \mapsto c_1$ *and* $F_2 \models c_1 \mapsto c_2$ *then* $F_1 \diamond F_2 \models c_0 \mapsto c_2$.

We are then ready to state a non-interference result in the manner of [18]:

Theorem 1 (Adequacy of Commands). *If* $\vdash^{[q_t:c_t]}_{[q_s:c_s]} C : \mathsf{E},\mathsf{I}\,\&\,\chi$ *and* $\chi \rightsquigarrow \{q_t\}$ *and* $\mathcal{L}(q_t) = L$ *and* $\mathsf{fv}(c_s) \rightsquigarrow \{q_s\}$ *then we have* $[\![(\mathsf{E},\mathsf{I}[q_s \mapsto c_s][q_t \mapsto c_t]) : q_s \mapsto q_t]\!] \models c_s \mapsto c_t$.

Proof. We proceed by induction on $\vdash^{[q_t:c_t]}_{[q_s:c_s]} C : \mathsf{E},\mathsf{I}\,\&\,\chi$.

Case: Assignment. Assume that $(\sigma_0,\delta_0) \equiv_{c_s} (\sigma_0',\delta_0')$ and that

$$\gamma \in [\![(\{(q_s, g \to \boldsymbol{x} := \boldsymbol{e}\colon \boldsymbol{r}, q_t)\}), [q_s \mapsto c_s][q_t \mapsto c_t]) : q_s \mapsto q_t]\!](\sigma_0,\delta_0)$$

In case $\gamma = (\sigma_1,\delta_1)$ it follows that there exists $d \geq 0$ such that $\sigma_1 = [\![\boldsymbol{x} := \boldsymbol{e}]\!]\sigma_0$ and $\delta_1 = (\delta_0 + d)[\boldsymbol{r} \mapsto \boldsymbol{0}]$, and such that $[\![g]\!](\sigma_0,(\delta_0+d)) = \mathsf{tt}$, $[\![c_s]\!](\sigma_0,\delta_0+d) = \mathsf{tt}$ and $[\![c_t]\!](\sigma_1,\delta_1) = \mathsf{tt}$. Defining $\gamma' = (\sigma_1',\delta_1') = ([\![\boldsymbol{x} := \boldsymbol{e}]\!]\sigma_0', (\delta_0' + d)[\boldsymbol{r} \mapsto \boldsymbol{0}])$ ensures that $[\![g]\!](\sigma_0',(\delta_0' + d)) = \mathsf{tt}$, $[\![c_s]\!](\sigma_0',\delta_0' + d) = \mathsf{tt}$ and $[\![c_t]\!](\sigma_1',\delta_1') = \mathsf{tt}$ because all variables and clocks tested are low and hence

$$\gamma \equiv_{c_t} \gamma' \in [\![(\{(q_s, g \to \boldsymbol{x} := \boldsymbol{e}\colon \boldsymbol{r}, q_t)\}), [q_s \mapsto c_s][q_t \mapsto c_t]) : q_s \mapsto q_t]\!](\sigma_0',\delta_0')$$

In case $\gamma = \bot$ it follows that there is no value of $d \geq 0$ such that $[\![g]\!](\sigma_0,(\delta_0+d)) = \mathsf{tt}$, $[\![c_s]\!](\sigma_0,\delta_0+d) = \mathsf{tt}$ and $[\![c_t]\!](\sigma_1,\delta_1) = \mathsf{tt}$. Then there also is no value of $d \geq 0$ such that $[\![g]\!](\sigma_0',(\delta_0'+d)) = \mathsf{tt}$, $[\![c_s]\!](\sigma_0',\delta_0'+d) = \mathsf{tt}$ and $[\![c_t]\!](\sigma_1',\delta_1') = \mathsf{tt}$ because all variables and clocks tested are low and hence setting $\gamma' = \bot$ establishes that

$$\gamma \equiv \gamma' \in [\![(\{(q_s, g \to \boldsymbol{x} := \boldsymbol{e}\colon \boldsymbol{r}, q_t)\}), [q_s \mapsto c_s][q_t \mapsto c_t]) : q_s \mapsto q_t]\!](\sigma_0',\delta_0')$$

The other direction is similar and this completes the assignment case.

Case: Publish. This case is analogous to the case for assignment.

Case: Sequence. We shall write

$$F = [\![(\mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{I}_1 \cup \mathsf{I}_2[q_s \mapsto c_s][q \mapsto c][q_t \mapsto c_t]) : q_s \mapsto q_t]\!]$$
$$F_1 = [\![(\mathsf{E}_1, \mathsf{I}_1[q_s \mapsto c_s][q \mapsto c]) : q_s \mapsto q]\!]$$
$$F_2 = [\![(\mathsf{E}_2, \mathsf{I}_2[q \mapsto c][q_t \mapsto c_t]) : q \mapsto q_t]\!]$$

and observe that $F = F_1 \diamond F_2$. The result then follows from the induction hypotheses and Fact 2.

Case: Looping. We shall write

$$F = [\![(\bigcup_i \mathsf{E}_i, \bigcup_i \mathsf{I}_i[q_s \mapsto c_s][q_t \mapsto c_t]) : q_s \mapsto q_t]\!]$$
$$F_i = \begin{cases} [\![(\mathsf{E}_i, \mathsf{I}_i[q_s \mapsto c_s]) : q_s \mapsto q_s]\!] & \text{whenever } i \leq n \\ [\![(\mathsf{E}_i, \mathsf{I}_i[q_s \mapsto c_s][q_t \mapsto c_t]) : q_s \mapsto q_t]\!] & \text{whenever } i > n \end{cases}$$

and this gives rise to the equation

$$F = (\bigcup_{i=1}^{n} F_i \diamond F) \ \cup \bigcup_{i=n+1}^{m} F_i$$

We shall consider two subcases, one where the condition $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}]$ is true and one where it is false.

Subcase: Looping when $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}]$ *is true.* In this case (using the notation of Fig. 3) all the variables and clocks in $\bigcup_{i=1}^{m} \mathsf{fv}(\mathsf{fst}^{c_i}_{c_s}(T_i))$ are low. Assume that $(\sigma_0, \delta_0) \equiv_{c_s} (\sigma'_0, \delta'_0)$ and that $\gamma \in F(\sigma_0, \delta_0)$. This must be because of a trace as considered in Sect. 2.

If this trace visits q_s infinitely often we will be able to construct a sequence $k_1, k_2, \cdots, k_i, \cdots$ such that each $k_i \leq n$ and

$$\forall i > 0 : (\sigma_i, \delta_i) \in F_{k_i}(\sigma_{i-1}, \delta_{i-1})$$

and $\gamma = \bot$. By the induction hypothesis we can find (σ'_i, δ'_i) such that

$$\forall i > 0 : (\sigma_i, \delta_i) \equiv_{c_s} (\sigma'_i, \delta'_i) \in F_{k_i}(\sigma'_{i-1}, \delta'_{i-1})$$

and this establishes that $\bot \in F(\sigma'_0, \delta'_0)$.

If the trace visits q_s only finitely often we will be able to construct a sequence $k_1, k_2, \cdots, k_j$ such that $\forall i < j : k_i \leq n$ and $k_j \leq m$ and

$$\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \in F_{k_i}(\sigma_{i-1}, \delta_{i-1})$$
$$\gamma \in F_{k_j}(\sigma_{j-1}, \delta_{j-1})$$

By the induction hypothesis we can find (σ'_i, δ'_i) and γ' such that

$$\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \equiv_{c_s} (\sigma'_i, \delta'_i) \in F_{k_i}(\sigma'_{i-1}, \delta'_{i-1})$$
$$\gamma \equiv \gamma' \in F_{k_j}(\sigma'_{j-1}, \delta'_{j-1})$$

and this establishes that $\gamma' \in F(\sigma'_0, \delta'_0)$.

The other direction is similar and this completes the subcase.

Subcase: Looping when $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[^{q_t:c_t}_{q_s:c_s}]$ *is false.* In this case all the variables and clocks in $\bigcup_{i=1}^{n+1} \mathsf{fv}(\mathsf{fst}^{c_i}_{c_s}(T_i))$ are low but this is not necessarily the case for those in $\bigcup_{i=n+1}^{m} \mathsf{fv}(\mathsf{fst}^{c_i}_{c_s}(T_i))$; however, we do know that $[\![c_s]\!](\sigma, \delta) \Rightarrow \bot \notin F(\sigma, \delta)$.

Assume that $(\sigma_0, \delta_0) \equiv_{c_s} (\sigma'_0, \delta'_0)$ and that $\gamma \in F(\sigma_0, \delta_0)$. The assumptions of the subcase ensure that $\gamma \neq \bot$.

We will be able to construct a sequence $k_1, k_2, \cdots, k_j$ such that $\forall i < j : k_i \leq n$ and $k_j > n$ and

$$\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \in F_{k_i}(\sigma_{i-1}, \delta_{i-1})$$
$$\gamma \in F_{k_j}(\sigma_{j-1}, \delta_{j-1})$$

By the induction hypothesis we can find (σ'_i, δ'_i) such that

$$\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \equiv_{c_s} (\sigma'_i, \delta'_i) \in F_{k_i}(\sigma'_{i-1}, \delta'_{i-1})$$

There are now two scenarios for how to proceed.

Subcase scenario where all variables and clocks in $\mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T_{k_j}))$ *are low.* In this case we can find $\gamma' \in F_{k_j}(\sigma'_{j-1}, \delta'_{j-1})$ such that $\gamma \equiv \gamma'$.

Subcase scenario where at least one variable or clock in $\mathsf{fv}(\mathsf{fst}^{c_t}_{c_s}(T_{k_j}))$ *is high.* Then $\mathsf{ass}(T_{k_j})$ cannot contain any low variable or clock and hence there is $d \geq 0$ such that $\gamma \equiv (\sigma_{j-1}, \delta_{j-1} + d)$ where the addition of d takes care of the potential delay in q_s. Next we use that $\bot \notin F(\sigma'_{j-1}, \delta'_{j-1})$ to obtain $k'_j, \sigma'_j, \delta'_j$ such that $(\sigma'_j, \delta'_j) \in F_{k'_j}(\sigma'_{j-1}, \delta'_{j-1})$.

It cannot be the case that $k'_j \leq n$. To see this, assume by way of contradiction that $k'_j \leq n$. Then $(\sigma_{j-1}, \delta_{j-1})$ would be a witness for $\underline{\mathsf{sat}}(\mathsf{fst}^{c_t}_{c_s}(T_{k_j}) \wedge \mathsf{fst}^{c_s}_{c_s}(T_{k'_j}))$ ensuring that $\mathsf{fst}^{c_t}_{c_s}(T_{k_j}) \rightsquigarrow \mathsf{ass}(T_{k'_j})$ so that $\mathsf{ass}(T_{k'_j})$ could not contain a low variable or clock. It would follow that there would be $d' \geq 0$ such that $(\sigma'_j, \delta'_j) \equiv_{c_s} (\sigma_{j-1}, \delta_{j-1} + d')$ where the addition of d' is due to the possibility of delay in q_s. But then we would be able to construct an infinite sequence (σ'_l, δ'_l) for $l > j$ such that $(\sigma'_l, \delta'_l) \in F_{k'_j}(\sigma'_{l-1}, \delta'_{l-1})$ and $(\sigma'_l, \delta'_l) \equiv_{c_s} (\sigma'_{j-1}, \delta'_{j-1} + d'$ would hold for $l \geq j$. But this would contradict the fact that $\bot \notin F(\sigma'_j, \delta'_j)$.

We are left with the case where $k'_j > n$. We must have that $\mathsf{ass}(T_{k'_j})$ cannot contain any low variable or clock: either one variable or clock in $\mathsf{fst}^{c_t}_{c_s}(T_{k'_j})$ is high and it follows as in a case above, or all variables and clocks in $\mathsf{fst}^{c_t}_{c_s}(T_{k'_j})$ are low and it follows because $(\sigma_{j-1}, \delta_{j-1} + d)$ is a witness for $\underline{\mathsf{sat}}(\mathsf{fst}^{c_t}_{c_s}(T_{k_j}) \wedge \mathsf{fst}^{c_s}_{c_s}(T_{k'_j}))$ and we could proceed as in a case above. Hence $(\sigma'_j, \delta'_j) = (\sigma'_{j-1}, \delta'_{j-1} + d')$ for some $d' \geq 0$.

It remains to show that d' can be chosen to be d. For this we use that all clocks in $\mathsf{fst}^{c_t}_{c_s}(T_{k_j})$ and $\mathsf{fst}^{c_s}_{c_s}(T_{k'_j})$ are low and that $\overline{\mathsf{fst}^{c_t}_{c_s}(T_{k_j})} = \overline{\mathsf{fst}^{c_s}_{c_s}(T_{k'_j})}$.

The other direction is similar and this completes the subcase.

We can now establish our main result that the type system enforces a sufficient condition for the absence of information flows violating the security policy.

Corollary 1 (Adequacy). *If* $\vdash \mathtt{begin}^{[c_\circ]}\, C\, {}^{[c_\bullet]}\mathtt{end} : \mathsf{E}, \mathsf{I}, q_\circ, q_\bullet$ *then we have that* $[\![(\mathsf{E}, \mathsf{I}) : q_\circ \mapsto q_\bullet]\!] \models \mathsf{I}(q_\circ) \mapsto \mathsf{I}(q_\bullet)$.

7 Conclusion

We have shown how to successfully merge Timed Automata with Information Flow and Language Based Security through the introduction of the Timed Commands language patterned after Dijkstra's Guarded Commands. This has facilitated developing a type system that prevents unnecessary *label creep* and that deals with *non-determinism*, *non-termination* and *continuous real-time*. The type system has been proved adequate by means of a non-interference result (with observable non-determinism).

We are exploring how to automate the analysis and in particular how to implement (a sound approximation of) the $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}$ predicate indicating the lack of termination of the looping construct. One possible way, is to use existing methodologies that deal with *time-lock* (deadlock) freedom checks for timed automata. The check of the predicate $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}$ then amounts to check for *time-lock* freedom (infinite loops) or *time-locks* that do not occur at the final nodes (stack configurations) of the particular loop construct that the $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}$ predicate refers too. The work of [8] presents a tool which is used in the conjuction with UPPAAL and is able to detect possible sources of deadlocks in timed-automata.

We are considering how to deal with more concepts from Timed Automata as for example urgents actions. Our treatment of `publish` $\boldsymbol{e}$ could be extended to a more general treatment of declassification and endorsement as permitted in the Decentralized Label Model [14]; our flow based security condition should suffice for expressing semantic correctness. To strengthen the security policies that can be expressed we are contemplating incorporating the content-dependent policies of [15].

A longer term goal is to allow policies to simultaneously dealing with safety and security properties of cyberphysical systems.

Acknowledgment. The authors are supported in part by the IDEA4CPS Reseearch Centre studying the Foundations for Cyber-Physical Systems and granted by the Danish Research Foundation for Basic Research (DNRF86-10). We would like to thank Ximeng Li for commenting upon a previous version.

References

1. Aceto, L., Ingolfsdottir, A., Larsen, K.G., Srba, J.: Reactive Systems: Modelling, Specification and Verification. Cambridge University Press, Cambridge (2007)
2. Agat, J.: Transforming out timing leaks. In: Proceedings of the POPL, pp. 40–53 (2000)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
4. Apt, K.R.: Ten years of Hoare's logic: a survey - part 1. ACM Trans. Program. Lang. Syst. **3**(4), 431–483 (1981)
5. Banâtre, J.-P., Bryce, C., Métayer, D.: Compile-time detection of information flow in sequential programs. In: Gollmann, D. (ed.) ESORICS 1994. LNCS, vol. 875, pp. 55–73. Springer, Heidelberg (1994). doi:10.1007/3-540-58618-0_56

6. Barbuti, R., De Francesco, N., Santone, A., Tesei, L.: A notion of non-interference for timed automata. Fundam. Inform. **51**(1–2), 1–11 (2002)
7. Barbuti, R., Tesei, L.: A decidable notion of timed non-interference. Fundam. Inform. **54**(2–3), 137–150 (2003)
8. Bordbar, B., Okano, K.: Testing deadlock-freeness in real-time systems: a formal approach. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, pp. 95–109. Springer, Heidelberg (2005). doi:10.1007/978-3-540-31848-4_7
9. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Commun. ACM **20**(7), 504–513 (1977)
10. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM **18**(8), 453–457 (1975)
11. Focardi, R., Gorrieri, R., Martinelli, F.: Real-time information flow analysis. IEEE J. Sel. Areas Commun. **21**(1), 20–35 (2003)
12. Gardey, G., Mullins, J., Roux, O.H.: Non-interference control synthesis for security timed automata. Electr. Notes Theor. Comput. Sci. **180**(1), 35–53 (2007)
13. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Time and probability-based information flow analysis. IEEE Trans. Softw. Eng. **36**(5), 719–734 (2010)
14. Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: ACM Symposium on Operating System Principles, SOSP 1997, pp. 129–142. ACM (1997)
15. Hanne Riis Nielson and Flemming Nielson: Content dependent information flow control. J. Log. Algebr. Meth. Program. **87**, 6–32 (2017)
16. UPPAAL. http://www.uppaal.com/index.php?sida=200&rubrik=95
17. Volpano, D.M., Smith, G., Irvine, C.E.: A sound type system for secure flow analysis. J. Comput. Secur. **4**(2/3), 167–188 (1996)
18. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: Proceedings of the CSFW, pp. 29–43 (2003)