Lifting Verification Results for Preemption Statements

Manuel Gesell, Andreas Morgenstern, and Klaus Schneider

Embedded Systems Group, Department of Computer Science,
University of Kaiserslautern, Germany

Abstract. The normal operation of synchronous modules may be temporarily suspended or finally aborted due to requests of their environment. Hence, if a temporal logic specification has already been verified for a synchronous module, then the available verification result can typically only be used if neither suspension nor abortion will take place. Also, the simulation of synchronous modules has to be finally aborted so that temporal logic specifications referring to infinite behaviors cannot be completely answered. In this paper, we therefore define transformations on temporal logic specifications to lift available verification results for synchronous modules without suspension or abortion to refined temporal logic specifications that take care of these preemption statements. This way, one can establish simulation and modular verification of synchronous modules in contexts where preemptions are used.

1 Introduction

Reactive systems have been introduced as a special class of systems that have an ongoing interaction with their environment [11]. Their execution is divided into reaction steps, where the system reads inputs from the environment and reacts by computing the corresponding outputs. In contrast to interactive systems, the environment is allowed to initiate the interactions at any time, so that reactive systems usually have to work under real-time constraints. Typical examples are synchronous hardware circuits, many protocols, and many embedded and cyber-physical systems.

For the design of reactive systems, synchronous languages have been developed [9,3] whose paradigm directly reflects the reactive nature of the systems they describe. In addition to the explicit notion of reaction steps, languages like Esterel [4] and Quartz [15] offer many convenient statements for the design of reactive systems. One class of such statements are preemption statements for abortion and suspension that overwrite the normal behavior of the system when a specified condition holds. For example, the abortion statement ***abort*** S ***when***(σ) behaves as its body statement S as long as the condition σ is false, and immediately terminates when σ holds. The suspension statement ***suspend*** S ***when***(σ) also behaves as its body statement S as long as the condition σ is false, and suspends the computation in each step where σ holds. Both preemption statements can moreover be weak or strong which makes a difference on their influence on the control and data flow of the controlled statement S: While the weak versions allow the data flow actions to take place even if the condition σ holds, the strong versions also block the data flow.

Since reactive systems are often used in safety-critical applications, their functional correctness is of essential importance. For this reason, simulation and formal verification are routine steps in their design flows, and in particular, model checking is often used for these systems. However, due to the well-known state space explosion problem, a modular or compositional verification [7,6] is desired where modules can be replaced by their already verified properties. Large reactive systems can only be verified by modular or compositional approaches despite the tremendous progress on model checking procedures we have seen in the past two decades. Another reason for modular verification is that modules are defined for being reused later on, and therefore the effort for formal verification amortizes when one can simply reuse also the already verified properties.

However, it is clear that calling a module S in a preemption statement changes the behavior, so that temporal properties that hold for S may no longer be valid for the entire statement. It is therefore unclear how one can reuse available verification results for the statement S, which leads to the central question answered by this paper: 'What can we say about temporal properties of ***(weak)abort*** S ***when***(σ) or ***(weak)suspend*** S ***when***(σ), when we know that S satisfies a temporal property φ?'

In this paper, we therefore define transformations to map a temporal logic formula φ to modified temporal logic formulas $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma)$, $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi,\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{wk}}(\varphi,\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{st}}(\varphi,\sigma)$ such that these formulas hold for ***weak abort*** S ***when***(σ), ***abort*** S ***when***(σ), ***weak suspend*** S ***when***(σ), and ***suspend*** S ***when***(σ), respectively, provided that S satisfies φ. It is clear that these formulas are equivalent to φ if σ is false, and that 'as much as possible' of φ should be retained.

The results we present in this paper are not only useful for modular verification, which is our main interest. In [2], the authors considered the problem to make specifications for the simulation of reactive systems, which is difficult since the simulation has to be aborted after some finite time, so that properties that refer to the infinite behavior of the system cannot be completely answered. Our results can be also used for simulation in preemption contexts.

In [8], we already established modular verification techniques for synchronous programs. There a preemption context was simulated by introducing new input variables for the verification task. Hence, some assumptions about the context were made during the verification of a module. In this paper, however, we lift a given verification result $\mathcal{M} \models \varphi$ where $\mathcal{M}$ does not consider any preemption statement to new results $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathcal{M},\sigma) \models \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma)$, $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\mathcal{M},\sigma) \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi,\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{wk}}(\mathcal{M},\sigma) \models \Theta_{\mathsf{sp}}^{\mathsf{wk}}(\varphi,\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{st}}(\mathcal{M},\sigma) \models \Theta_{\mathsf{sp}}^{\mathsf{st}}(\varphi,\sigma)$ where $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathcal{M},\sigma)$, $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\mathcal{M},\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{wk}}(\mathcal{M},\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{st}}(\mathcal{M},\sigma)$ are ***weak abort*** $\mathcal{M}$ ***when***(σ), ***abort*** $\mathcal{M}$ ***when***(σ), ***weak suspend*** $\mathcal{M}$ ***when***(σ), and ***suspend*** $\mathcal{M}$ ***when***(σ), respectively. Thus, concerning preemption statements, the results presented here are stronger since they allow us to introduce preemption in the module even if it has not been considered there from the beginning.

The outline of our paper is as follows: Section 2 explains the syntax and semantics of the linear temporal logic (LTL), the representation of synchronous systems by guarded actions and transition systems, and defines the preemptions $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathcal{G},\sigma)$, $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\mathcal{G},\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{wk}}(\mathcal{G},\sigma)$, and $\Theta_{\mathsf{sp}}^{\mathsf{st}}(\mathcal{G},\sigma)$ for a set of guarded actions $\mathcal{G}$. Then, in Section 3 the transformations $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma)$, $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi,\sigma)$, $\Theta_{\mathsf{sp}}^{\mathsf{wk}}(\varphi,\sigma)$, and $\Theta_{\mathsf{sp}}^{\mathsf{st}}(\varphi,\sigma)$ are defined and correctness proofs are given. Section 4 illustrates our approach.

2 Preliminaries

This section introduces the temporal logic LTL, the representation of synchronous systems by synchronous guarded actions, and their represented state transition systems as foundations for our transformations.

2.1 Syntax and Semantics of LTL

For specifications, we consider linear temporal logic, since it is well-known that branching time logics like CTL do not lend themselves well for modular verification [12]. Given a finite set of variables $\mathcal{V}$, the following grammar rules with starting symbol S define the formulas of the temporal logic LTL.

$$S ::= \mathsf{A}P \qquad P ::= 0 \mid 1 \mid \mathcal{V} \mid \neg P \mid P \wedge P \mid P \vee P \mid \mathsf{X}P \mid [P\ \underline{\mathsf{U}}\ P] \mid [P\ \mathsf{U}\ P]$$

The symbol S represents thereby state formulas and P represents the path formulas. Similar to preemption statements, $[\varphi\ \underline{\mathsf{U}}\ \psi]$ is often called the 'strong until' while $[\varphi\ \mathsf{U}\ \psi]$ is called the 'weak until' operator. It is well-known that these operators are sufficient to define LTL, but for convenience, we may also introduce further operators like $\mathsf{G}\varphi := [\varphi\ \mathsf{U}\ 0]$ (always), $\mathsf{F}\varphi := [1\ \underline{\mathsf{U}}\ \varphi]$ (eventual), $[\varphi\ \mathsf{W}\ \psi] := [\neg\psi\ \mathsf{U}\ \varphi \wedge \psi)]$ (weak when), and $[\varphi\ \underline{\mathsf{W}}\ \psi] := [\neg\psi\ \underline{\mathsf{U}}\ (\varphi \wedge \psi)]$ (strong when). Their meaning is defined on state transition systems.

Definition 1 (Transition Systems). *A transition system $\mathcal{T} = (\mathcal{S},\mathcal{I},\mathcal{R},\mathcal{L})$ for a finite set of variables $\mathcal{V}$ is given by a finite set of states $\mathcal{S} \subseteq 2^{\mathcal{V}}$, a set of initial states $\mathcal{I} \subseteq \mathcal{S}$, a transition relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$, and a label function $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{V}}$ that maps each state to the set of variables that hold in this state.*

An infinite path is a function $\pi : \mathbb{N} \to \mathcal{S}$ with $(\pi^{(t)}, \pi^{(t+1)}) \in \mathcal{R}$, where we denote the t-th state of the path π as $\pi^{(t-1)}$ for $t \in \mathbb{N}$. The semantics of path formulas of a transition system $\mathcal{T}$ is defined by the relation $(\mathcal{T},\pi,t) \models \varphi$ that defines if a path formula φ holds on position t of a path π of a transition system $\mathcal{T}$ (see e. g. [14] for a full definition).

- $(\mathcal{T},\pi,t) \models p$ holds iff $p \in \mathcal{L}(\pi^{(t)})$ for every $p \in \mathcal{V}$
- $(\mathcal{T},\pi,t) \models \mathsf{X}\varphi$ holds iff $(\mathcal{T},\pi,t+1) \models \varphi$
- $(\mathcal{T},\pi,t) \models [\varphi\ \underline{\mathsf{U}}\ \psi]$ holds iff there is a δ such that $(\mathcal{T},\pi,t+\delta) \models \psi$ and for all $x < \delta$, we have $(\mathcal{T},\pi,t+x) \models \varphi$
- $(\mathcal{T},\pi,t) \models [\varphi\ \mathsf{U}\ \psi]$ holds iff $(\mathcal{T},\pi,t) \models [\varphi\ \underline{\mathsf{U}}\ \psi]$ or for all x, we have $(\mathcal{T},\pi,t{+}x) \models \varphi$.

$\mathsf{A}\varphi$ holds in a state s of $\mathcal{T}$ if all paths π starting in s satisfy $(\mathcal{T},\pi,0) \models \varphi$. Finally, a transition system $\mathcal{T}$ satisfies a LTL formula $\mathsf{A}\Phi$ if all initial states satisfy Φ, in this case, we write $\mathcal{T} \models \mathsf{A}\Phi$.

2.2 The Synchronous Model of Computation

The execution of synchronous languages [9,3] is divided into a discrete sequence of reaction steps that are also called macro steps. Within each macro step, the system reads all inputs and instantaneously generates all outputs depending on the current state and the read inputs. Also, the next state is computed in parallel to the current outputs. There are many synchronous languages including Esterel [4], Quartz [15], Lustre [10], Signal [13], and SyncCharts [1]. In the following, we do not focus on a particular synchronous language, and therefore use synchronous guarded actions as an intermediate representation for any synchronous language. An example for generating guarded actions for a Quartz program is presented in [8] while [5,15] describes the general compilation.

Definition 2 (Synchronous Guarded Actions). *A synchronous system over input $\mathcal{V}_i$, label $\mathcal{V}_l$, state $\mathcal{V}_s$, and output variables $\mathcal{V}_o$ is defined by a set of guarded actions. A guarded action is thereby a pair $\gamma \Rightarrow \alpha$ consisting of a boolean condition γ called the trigger of the guarded action and its action α. Actions are either immediate assignments $\boldsymbol{x} = \tau$ or delayed assignments* ***next(x)*** *$= \tau$ where $x \in \mathcal{V}_l \cup \mathcal{V}_s \cup \mathcal{V}_o$.*

The intuitive meaning of synchronous guarded actions is a *state transition system* over the variables $\mathcal{V} := \mathcal{V}_i \cup \mathcal{V}_l \cup \mathcal{V}_s \cup \mathcal{V}_o$ (see Definition 1). A state s is thereby a valuation of variables to their respective values and the transition relation will be formally defined below. Intuitively, the meaning is that whenever the guard is true in a state s, the action is fired, which means that the corresponding equation must be true. In case of an immediate assignment $\boldsymbol{x} = \tau$ this means that in state s, variable x must have the same value as τ, and for a delayed assignment **next(x)** $= \tau$, it means that in all successor states s', variable x must have the value that τ has on s. Whenever there is no guarded action that determines the value of a variable, a default action takes place. This default reaction assigns a default value for event variables, and the previous value for memorized variables. Input and label variables are always event variables, while state and output variables may be event or memorized variables. Both kinds of variables are important for the convenient modeling of reactive systems.

Furthermore, we partition the set of guarded actions into *control and data flow* actions, which will be important for defining strong and weak preemptions.

Definition 3 (Control and Data Flow). *The control flow are guarded actions writing a label variable $\mathcal{V}_l$, while the data flow are guarded actions writing a state variable $\mathcal{V}_s$ or an output $\mathcal{V}_o$. We assume that guarded actions of the control flow have the form $\gamma \Rightarrow$* ***next***$(\ell) =$ ***true****.*

Label variables $\mathcal{V}_l$ correspond with places in the program where the control flow can rest between the macro steps, i.e., these labels denote places in the program code where a macro step ends and where another one starts. By construction, these labels are translated to boolean variables where only guarded actions as shown above are obtained. Since labels are event variables, they will be automatically reset to ***false*** if there is no assignment making them ***true***.

The above informal remarks lead to the following formal definition of a state transition system. The aim is to generate boolean formulas for the initial state condition and the transition relation that can be directly used for model checking. To this end, we first define some auxiliary functions.

Definition 4 (Reactions per Variable). *Assume that for a variable $\boldsymbol{x} \in \mathcal{V}$, we have the guarded actions $(\gamma_1, \boldsymbol{x} = \tau_1), \ldots, (\gamma_p, \boldsymbol{x} = \tau_p)$ with immediate and $(\chi_1, \texttt{next(x)} = \pi_1), \ldots, (\chi_q, \texttt{next(x)} = \pi_q)$ with delayed assignments. Then, we define the following boolean formulas over $\mathcal{V} \cup \mathcal{V}'$, where $\boldsymbol{v}' \in \mathcal{V}'$ represents the variable $\boldsymbol{v} \in \mathcal{V}$ in the next step/state and* $\mathsf{Initial}(\boldsymbol{x})$ *denotes the initial value of variable $\boldsymbol{x}$ that is 0 for integers and **false** for booleans. Additionally, we make use of the substitution $\langle\varphi\rangle_{\mathcal{V}}^{\mathcal{V}'}$ that replaces all occurrences of a variable $\boldsymbol{v} \in \mathcal{V}$ in φ by the corresponding variable $\boldsymbol{v}' \in \mathcal{V}'$.*

- $\mathsf{Default}(\boldsymbol{x}) := \begin{cases} \mathsf{Initial}(\boldsymbol{x}) & : \textit{if } \boldsymbol{x} \textit{ is an event variable} \\ \boldsymbol{x} & : \textit{if } \boldsymbol{x} \textit{ is a memorized variable} \end{cases}$
- $\mathsf{ImmActs}(\boldsymbol{x}) := \bigwedge_{j=1}^{p} (\gamma_j \rightarrow \boldsymbol{x} = \tau_j)$
- $\mathsf{DelActs}(\boldsymbol{x}) := \bigwedge_{j=1}^{q} (\chi_j \rightarrow \boldsymbol{x}' = \pi_j)$
- $\mathsf{InitDefActs}(\boldsymbol{x}) := \left(\bigwedge_{j=1}^{p} \neg\gamma_j\right) \rightarrow \boldsymbol{x} = \mathsf{Initial}(\boldsymbol{x})$
- $\mathsf{NextDefActs}(\boldsymbol{x}) := \left\langle \bigwedge_{j=1}^{p} \neg\gamma_j \right\rangle_{\mathcal{V}}^{\mathcal{V}'} \wedge \left(\bigwedge_{j=1}^{q} \neg\chi_j\right) \rightarrow \boldsymbol{x}' = \mathsf{Default}(\boldsymbol{x})$

We will use the above formulas to construct now an initial state condition $\mathcal{I}$ and the transition relation $\mathcal{R}$ of a transition system.

Definition 5 (Symbolic Representation of Systems). *For a synchronous system over the variables $\mathcal{V}$ consisting of input $\mathcal{V}_i$, label $\mathcal{V}_l$, state $\mathcal{V}_s$, and output variables $\mathcal{V}_o$, the transition system $\mathcal{T} := (\mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L})$ is defined by the states $\mathcal{S} = 2^{\mathcal{V}}$, $\mathcal{L}(s) := s$, the following initial state condition $\mathcal{I}$, and the state transition relation $\mathcal{R}$, where $\mathcal{V}_{\mathsf{write}} := \mathcal{V}_l \cup \mathcal{V}_s \cup \mathcal{V}_o$ denotes the writable variables.*

- $\mathcal{I} := \bigwedge_{\boldsymbol{x} \in \mathcal{V}_{\mathsf{write}}} \mathsf{ImmActs}(\boldsymbol{x}) \wedge \bigwedge_{\boldsymbol{x} \in \mathcal{V}_{\mathsf{write}}} \mathsf{InitDefActs}(\boldsymbol{x})$
- $\mathcal{R} := \bigwedge_{\boldsymbol{x} \in \mathcal{V}_{\mathsf{write}}} \mathsf{ImmActs}(\boldsymbol{x}) \wedge \bigwedge_{\boldsymbol{x} \in \mathcal{V}_{\mathsf{write}}} \mathsf{DelActs}(\boldsymbol{x}) \wedge \bigwedge_{\boldsymbol{x} \in \mathcal{V}_{\mathsf{write}}} \mathsf{NextDefActs}(\boldsymbol{x})$

Whenever one of the guards γ_i of an immediate assignment $\gamma_i \Rightarrow \boldsymbol{x} = \tau_i$ holds in the definition of $\mathcal{R}$, then the equation $\boldsymbol{x} = \tau_i$ must hold, since the assignment has an immediate effect. Analogously, if a guard χ_i of a delayed assignment $\chi_i \Rightarrow$ ***next(x)*** $= \pi_i$ holds, then the equation $\boldsymbol{x}' = \pi_i$ that defines the value for $\boldsymbol{x}$ in the next step must hold. The value of $\boldsymbol{x}$ is determined by the default action if no guard χ_i held in the previous step and no guard γ_i holds in the current step.

2.3 Preemption Statements

In the following, we describe the semantics of the four different preemption statements (***(weak)abort, (weak)suspend***) used in Quartz[1].

[1] We only consider the ***immediate*** variants of these statements in this paper that observe the preemption condition also in the first macro step of the statement while other variants omit the starting point of time. All results presented here can be easily transferred to the omitted delayed variants as well.

Definition 6 (Preemption of Synchronous Systems). *Given guarded actions $\mathcal{G}$ of a synchronous system over input $\mathcal{V}_i$, label $\mathcal{V}_l$, state $\mathcal{V}_s$, and output variables $\mathcal{V}_o$. Then, the weak/strong abortion and weak/strong suspension with a condition σ is obtained by modifying the guarded actions as follows to obtain synchronous systems $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathcal{G},\sigma)$, $\Theta^{\mathsf{wk}}_{\mathsf{ab}}(\mathcal{G},\sigma)$, $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\mathcal{G},\sigma)$, and $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\mathcal{G},\sigma)$, respectively.*

preemption		control flow $(\gamma \Rightarrow \textbf{next}(\ell) = \mathsf{true}) \in \mathcal{G}$	data flow $(\gamma \Rightarrow \alpha) \in \mathcal{G}$
strong abort σ	$\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathcal{G},\sigma)$	$\neg\sigma \wedge \gamma \Rightarrow \textbf{next}(\ell) = \mathsf{true}$	$\neg\sigma \wedge \gamma \Rightarrow \alpha$
weak abort σ	$\Theta^{\mathsf{wk}}_{\mathsf{ab}}(\mathcal{G},\sigma)$	$\neg\sigma \wedge \gamma \Rightarrow \textbf{next}(\ell) = \mathsf{true}$	$\gamma \Rightarrow \alpha$
strong suspend σ	$\Theta^{\mathsf{st}}_{\mathsf{sp}}(\mathcal{G},\sigma)$	$(\neg\sigma \wedge \gamma) \vee (\ell \wedge \sigma) \Rightarrow \textbf{next}(\ell) = \mathsf{true}$	$\neg\sigma \wedge \gamma \Rightarrow \alpha$
weak suspend σ	$\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\mathcal{G},\sigma)$	$(\neg\sigma \wedge \gamma) \vee (\ell \wedge \sigma) \Rightarrow \textbf{next}(\ell) = \mathsf{true}$	$\gamma \Rightarrow \alpha$

The table shows that the guarded actions of the data flow are only modified by the strong preemption statements since weak preemption allows data actions to take place at the time of preemption. Moreover, weak and strong abortions have the same effect on the control flow. Abortion statements disable all assignments to control flow labels ℓ so that the control flow leaves the system in case of abortion. During a suspension, the control flow is kept and does not move to other labels.

In addition, any preemption context represented by the transition system $\mathcal{T}' := (\mathcal{S}',\mathcal{I}',\mathcal{R}',\mathcal{L}')$ changes the behavior only if σ holds. Hence on a path π where no preemption takes place ($\forall i.\pi^{(i)} \not\models \sigma$), the behavior of $\mathcal{T}'$ is equivalent to the original transition system $\mathcal{T} := (\mathcal{S},\mathcal{I},\mathcal{R},\mathcal{L})$. Hence, it is clear that we have $\mathcal{S} \subseteq \mathcal{S}'$, $\mathcal{I} \subseteq \mathcal{I}'$ and $\mathcal{R} \subseteq \mathcal{R}'$, which allows us to apply the following lemma:

Lemma 1. *Let $\mathcal{T} = (\mathcal{S},\mathcal{I},\mathcal{R},\mathcal{L})$ and $\mathcal{T}' = (\mathcal{S}',\mathcal{I}',\mathcal{R}',\mathcal{L}')$ be two transition systems where $\mathcal{S} \subseteq \mathcal{S}'$, $\mathcal{I} \subseteq \mathcal{I}'$, $\mathcal{R} \subseteq \mathcal{R}'$, and $\mathcal{L}(\vartheta) = \mathcal{L}'(\vartheta)$ holds for any state $\vartheta \in \mathcal{S}$. Then, there exists a simulation relation $\preceq$ between $\mathcal{T}$ and $\mathcal{T}'$.*

Proof. Simply define the simulation relation $\preceq$ as follows: $\vartheta_1 \preceq \vartheta_2 :\Leftrightarrow \vartheta_1 = \vartheta_2$, i.e. $\preceq$ is the identity relation that satisfies the simulation relation properties. ∎

3 Making LTL Specifications Preemptive

In general, a temporal logic formula φ that holds in a synchronous system given by its guarded actions $\mathcal{G}$ will no longer be valid in one of the systems $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathcal{G},\sigma)$, $\Theta^{\mathsf{wk}}_{\mathsf{ab}}(\mathcal{G},\sigma)$, $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\mathcal{G},\sigma)$, and $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\mathcal{G},\sigma)$. For example, the system $\mathcal{G} = \{\mathsf{true} \Rightarrow \textbf{next}(\ell) = \mathsf{true}, \ell \Rightarrow c = i\}$ with $\mathcal{V}_i = \{i\}$, $\mathcal{V}_l = \{\ell\}$, and $\mathcal{V}_o = \{c\}$ is modified to $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathcal{G}, abrt) = \{\neg abrt \Rightarrow \textbf{next}(\ell) = \mathsf{true}, \neg abrt \wedge \ell \Rightarrow c = i\}$. Therefore, the LTL specification $\textbf{A}\ \textbf{G}\ (c \leftrightarrow i)$ that holds on $\mathcal{G}$ is no longer satisfied in $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathcal{G}, abrt)$. However, a specification like $\textbf{A}[(c \leftrightarrow i)\ \mathsf{U}\ abrt]$ holds, which states that c is equivalent to i until an abortion takes place.

In the following, we define transformations $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\varphi,\sigma)$, $\Theta^{\mathsf{wk}}_{\mathsf{ab}}(\varphi,\sigma)$, $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\varphi,\sigma)$, and $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\varphi,\sigma)$ for temporal logic formulas φ so that we establish the following modular proof rules. These rules allow us to reason about a satisfied temporal logic property (e.g. $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\varphi,\sigma)$) of a system in a preemption context (e.g. $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathcal{G},\sigma)$), in case the property φ has already been proved for $\mathcal{G}$. Since we want to use our rules in an interactive verification tool that considers systems defined by guarded actions, we define these rules directly on guarded actions. Nevertheless, the correctness proofs will use the equivalent representation of transition systems that we defined in the previous section.

$$\frac{\mathcal{G} \models \varphi}{\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathcal{G},\sigma) \models \Theta^{\mathsf{st}}_{\mathsf{ab}}(\varphi,\sigma)} \qquad \frac{\mathcal{G} \models \varphi}{\Theta^{\mathsf{wk}}_{\mathsf{ab}}(\mathcal{G},\sigma) \models \Theta^{\mathsf{wk}}_{\mathsf{ab}}(\varphi,\sigma)}$$

$$\frac{\mathcal{G} \models \varphi \quad \mathsf{DFNxtEvtFree}(\mathcal{G})}{\Theta^{\mathsf{st}}_{\mathsf{sp}}(\mathcal{G},\sigma) \models \Theta^{\mathsf{st}}_{\mathsf{sp}}(\varphi,\sigma)} \qquad \frac{\mathcal{G} \models \varphi}{\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\mathcal{G},\sigma) \models \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\varphi,\sigma)}$$

The upper part defines the assumptions of the rule, the lower part defines the conclusions that hold by the rule. The condition $\mathsf{DFNxtEvtFree}(\mathcal{G})$ and the transformation $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\varphi,\sigma)$ are explained in Section 3.2.

To this end, we assume without loss of generality that the given specification φ is in negation normal form and the next operators are shifted inwards such that next operators only occur in front of a variable, its negation or a next operator.

3.1 Transformation for Strong Abortion

An abortion can stop the execution of a system in every step. Hence, a preemptive specification should express that either the specification φ has already been satisfied or that the execution was aborted in a step before the specification was fulfilled (or violated). These thoughts lead to the following definition.

Definition 7 (Transformation $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\varphi,\sigma)$). *The transformation $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\varphi,\sigma)$ that generates an* ***abort****-sensitive specification for* $\mathsf{A}\varphi$ *is defined recursively as*

$$\Theta^{\mathsf{st}}_{\mathsf{ab}}(\varphi,\sigma) := \begin{cases} \sigma \vee \varphi, & \text{if } \varphi \text{ is propositional} \\ \sigma \vee \boldsymbol{X}(\Theta^{\mathsf{st}}_{\mathsf{ab}}(\psi,\sigma)), & \text{if } \varphi = \boldsymbol{X}\psi \\ [\Theta^{\mathsf{st}}_{\mathsf{ab}}(\psi,\sigma) \otimes \Theta^{\mathsf{st}}_{\mathsf{ab}}(\gamma,\sigma)], & \text{if } \varphi = \psi \otimes \gamma \text{ with } \otimes \in \{\underline{\mathsf{U}}, \mathsf{U}\} \\ \Theta^{\mathsf{st}}_{\mathsf{ab}}(\psi,\sigma) \otimes \Theta^{\mathsf{st}}_{\mathsf{ab}}(\gamma,\sigma), & \text{if } \varphi = \psi \otimes \gamma \text{ with } \otimes \in \{\wedge, \vee\}. \end{cases}$$

The crucial point of the definition is that we have to forbid the use of a variable after an abortion took place, which is achieved in that all recursive calls will finally introduce a disjunction with σ. The definition states that for the next operator, the specification $\varphi = \boldsymbol{X}\psi$ must lead to a specification that requires that the execution is aborted in the current or next step since σ holds or ψ holds in the next step. Thus, the specification $\varphi := [\psi \ \mathsf{U}\ \gamma]$ (and $[\psi\ \underline{\mathsf{U}}\ \gamma]$ respectively) requires that ψ holds in every step until (eventually) γ or σ holds (the condition σ is added implicitly by the recursive calls). Note that it is impossible to abort the left-hand side of a (strong) until without aborting the right-hand side, too. The same is valid for the Boolean operators because σ is added simultaneously on both sides. For a propositional formula φ, we have for example $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathsf{G}\varphi,\sigma) = [\varphi\ \mathsf{U}\ \sigma]$ and $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\mathsf{F}\varphi,\sigma) = \mathsf{F}(\varphi \vee \sigma)$.

Correctness. To prove the correctness of the proof rule related to the above transformation, we will make use of the following lemmata.

Lemma 2 (Containment of φ). *The transformation preserves the original specification if no preemption takes place, i.e., $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \mathsf{false}) = \varphi$ holds.*

Proof. The lemma can be easily proved by induction over φ. ■

The following lemma states that the transformed specifications are vacuously satisfied if σ holds.

Lemma 3. *For an arbitrary but fixed condition σ and a path π' through $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\mathcal{G}, \sigma)$ and a position m such that $\pi'^{(m)} \vdash \sigma$ holds, we have*

$$(\Theta_{\mathsf{ab}}^{\mathsf{st}}(\mathcal{G}, \sigma), \pi', m) \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \sigma).$$

Proof. The proof can be easily shown by an induction over the structure of φ.

The following theorem ensures the correctness of the modular proof rule for strong abortion, and even that the assumption and conclusion of the rule are equivalent.

Theorem 1. *For any set of guarded actions $\mathcal{G}$ and any condition σ, the following holds*

$$\Theta_{\mathsf{ab}}^{\mathsf{st}}(\mathcal{G}, \sigma) \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \sigma) \leftrightarrow \mathcal{G} \models \varphi.$$

Proof. The '$\rightarrow$' direction states that we retained 'as much as possible' in our transformation and it follows directly from Lemma 2 and Lemma 1. The '$\leftarrow$' direction directly proves the correctness of the proof rule.

Let $\mathcal{T}$ be the original transition system for $\mathcal{G}$ and $\mathcal{T}'$ be the transition system for $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\mathcal{G}, \sigma)$. Obviously, if σ does not occur on a path π' through $\mathcal{T}'$, then the original system $\mathcal{T}$ already contained π' and we can conclude from Lemma 2 that $(T', \pi') \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \sigma) = \varphi$.

Assume we have a path $\pi \in \mathcal{T}$ through the original system and $\pi' \in \mathcal{T}'$ is a path that is equivalent to π up to a minimal position t_σ where σ holds. We show by finite induction on the number of temporal operators ($\|\varphi\|$) in an arbitrary formula φ, that $\forall m \leq t_\sigma$: if $(\mathcal{T}, \pi, m) \models \varphi$ we have $(\mathcal{T}', \pi', m) \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \sigma)$.

Base Case: $\|\varphi\| = 0$, hence φ is propositional and $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \sigma)$ is equivalent to $\varphi \vee \sigma$. A case distinction for $\pi'^{(m)}$ solves the case: for $\pi'^{(m)} \vdash \sigma$ we have $(\mathcal{T}', \pi', m) \models \sigma$ and for $\pi'^{(m)} \nvdash \sigma$ we have $(\mathcal{T}', \pi', m) \models \varphi$ following from the definition of π and π'. Hence, $(\mathcal{T}', \pi', m) \models \varphi \vee \sigma = \Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \sigma)$ holds.

Inductive Step: $\|\varphi\| = m + 1$, hence, $\Theta_{\mathsf{ab}}^{\mathsf{st}}(\varphi, \sigma)$'s result is besides the trivial boolean combinations either $\sigma \vee \boldsymbol{X}\Theta_{\mathsf{ab}}^{\mathsf{st}}(\psi, \sigma)$, $[\Theta_{\mathsf{ab}}^{\mathsf{st}}(\psi, \sigma)\ \underline{\mathsf{U}}\ \Theta_{\mathsf{ab}}^{\mathsf{st}}(\gamma, \sigma)]$, or $[\Theta_{\mathsf{ab}}^{\mathsf{st}}(\psi, \sigma)\ \mathsf{U}\ \Theta_{\mathsf{ab}}^{\mathsf{st}}(\gamma, \sigma)]$.

For the next operator we have $(\mathcal{T}, \pi, m) \models \boldsymbol{X}\psi \overset{def}{\Rightarrow} (\mathcal{T}, \pi, m+1) \models \psi$. If $m + 1 < t_\sigma$, we can apply the inductive hypothesis to conclude $(\mathcal{T}', \pi', m + 1) \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\psi, \sigma)$. Otherwise, σ holds at position $m + 1$, and one can conclude from Lemma 3 that $(\mathcal{T}', \pi', m + 1) \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\psi, \sigma)$

Now we turn to the strong-until-operator, i. e. we consider the case that $(\mathcal{T}, \pi, m) \models [\psi\ \underline{\mathsf{U}}\ \gamma]$, hence there exists a t_γ such that $\forall m \leq t' < t_\gamma$. $(\mathcal{T}, \pi, t') \models \psi$ and $(\mathcal{T}, \pi, t_\gamma) \models \gamma$. Hence, if $t_\gamma < t_\sigma$, we can use our inductive hypothesis to conclude that $\forall m \leq t' < t_\gamma.(\mathcal{T}', \pi', t') \models \Theta_{\mathsf{ab}}^{\mathsf{st}}(\psi, \sigma)$

and $(\mathcal{T}', \pi', t_\gamma) \models \Theta^{\mathsf{st}}_{\mathsf{ab}}(\gamma, \sigma)$. For the other case, we apply Lemma 3 to conclude that $(\mathcal{T}', \pi', t_\sigma) \models \Theta^{\mathsf{st}}_{\mathsf{ab}}(\gamma, \sigma)$ and we can apply the I.H. to prove that $\forall m \leq t' < t_\sigma.(\mathcal{T}', \pi', t') \models \Theta^{\mathsf{st}}_{\mathsf{ab}}(\psi, \sigma)$. Hence $(\mathcal{T}', \pi', m) \models \Theta^{\mathsf{st}}_{\mathsf{ab}}([\psi\ \underline{\mathsf{U}}\ \gamma], \sigma)$ holds in both cases. The case for weak until is shown analogously. ■

3.2 Transformation for Strong Suspension

A suspension can postpone the current execution of the guarded actions to a later point of time. Hence, no guarded action is executed during the suspension, but the delayed assignments of the previous step still take place. The ***suspend***-sensitive specification must ensure that either the execution of the system is suspended, and a violation of the specification is secondary (because no step of the original system is executed) or the next macro step of the system is executed, and as a consequence, the specification must be satisfied for this step. Note that it is possible to suspend the system infinitely often and that this case must be covered as well.

Unfortunately, the transformation defined below is not applicable if the *data flow* contains ***next*** assignments to *event* variables, because such an assignment may get lost during a suspension. The problem is explained in detail in Theorem 2. Hence, we exclude systems violating this requirement by adding the assumption $\mathsf{DFNxtEvtFree}(\mathcal{G})$ to the rule. This condition checks that the *data flow* is free of ***next*** assignments to *event* variables.

Definition 8 (Transformation $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\varphi, \sigma)$). *For a given specification $\mathsf{A}\varphi$, the transformation $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\varphi, \sigma)$ is defined as*

$$\Theta^{\mathsf{st}}_{\mathsf{sp}}(\varphi, \sigma) := \begin{cases} [\varphi\ \mathsf{W}\ \neg\sigma], & \text{if } \varphi \text{ is propositional} \\ \left[(\boldsymbol{X}\Theta^{\mathsf{st}}_{\mathsf{sp}}(\psi, \sigma))\ \mathsf{W}\ \neg\sigma\right], & \text{if } \varphi = \boldsymbol{X}\psi \\ [\Theta^{\mathsf{st}}_{\mathsf{sp}}(\psi, \sigma) \otimes \Theta^{\mathsf{st}}_{\mathsf{sp}}(\gamma, \sigma)], & \text{if } \varphi = \psi \otimes \gamma \text{ with } \otimes \in \{\underline{\mathsf{U}}, \mathsf{U}\} \\ \Theta^{\mathsf{st}}_{\mathsf{sp}}(\psi, \sigma) \otimes \Theta^{\mathsf{st}}_{\mathsf{sp}}(\gamma, \sigma), & \text{if } \varphi = \psi \otimes \gamma \text{ with } \otimes \in \{\wedge, \vee\}. \end{cases}$$

The crucial point is again that we have to forbid the use of a variable whenever the suspension takes place. Note again that all recursive calls will finally introduce a weak when operator. A module satisfying a specification $\varphi := \boldsymbol{X}\psi$ is suspendable in two macro steps. The definition states that the evaluation is postponed to the first point of time where σ becomes false. Thus, the specifications $\varphi := [\psi\ \mathsf{U}\ \gamma]$ (and $[\psi\ \underline{\mathsf{U}}\ \gamma]$ respectively) must lead to a specification that requires that ψ holds in every step until (eventually) γ holds or an (in)finite suspension takes place (covered by the weak when operator introduced by recursive calls).

We have $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\mathsf{G}\varphi, \sigma) = \mathsf{G}\,[\varphi\ \mathsf{W}\ \neg\sigma]$ and $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\mathsf{F}\varphi, \sigma) = \mathsf{F}\,[\varphi\ \mathsf{W}\ \neg\sigma]$, for a propositional φ.

An interesting fact is that an infinite suspension is equivalent to an abortion, hence only a special case of it. Hence, the transformation for abort can be also obtained from the suspension transformation.

Correctness. The following theorem ensures the correctness of the modular proof rule for strong suspension.

Theorem 2. *For any set of guarded actions $\mathcal{G}$, where* $\mathsf{DFNxtEvtFree}(\mathcal{G})$ *holds for $\mathcal{G}$ and any condition σ, we have $\Theta^{\mathsf{st}}_{\mathsf{sp}}(\mathcal{G},\sigma) \models \Theta^{\mathsf{st}}_{\mathsf{sp}}(\varphi,\sigma) \leftrightarrow \mathcal{G} \models \varphi$.*

Since the already proved rule for ***abort*** is a special case of the suspension rule, we only have to extend the proof of Theorem 1 at the appropriate places. We will omit this here and only describe the proof idea with help of Figure 1. There, the effect of a suspension on a simple Quartz program (given in Figure 2) is described in Figure 1. We consider three important points of time t_0, t_1 and t_{1s}: t_0 corresponds to a not suspended macro step starting in l_0, where the next assignment to $\boldsymbol{x}$ takes place. The time step t_1 is the intended execution of the macro step starting in l_1, but this step is now suspended. Nevertheless, the assignment to the variable $\boldsymbol{x}$ from the previous step takes place (v_0), but the immediate assignment to $\boldsymbol{y}$ is postponed until t_{1s}, which is the first point of time where the suspension is released. The assertion $\varphi(x,y)$ intended to be evaluated at point t_1 is postponed as well. It is no problem to evaluate $\varphi(x,y)$ in t_{1s}, since the immediate assignment is executed in the same step and for the delayed assignment the default reaction transfers the value v_0 to the step t_{1s} (indicated by the dashed box). Unfortunately, this holds only for memorized variables, since event variables are set to the type's default value and so the value v_0 gets lost during suspension. Hence, the example shows that a ***next*** assignment to an *event* variable in the data flow may completely change the behavior of the system. Hence, nothing can be deduced from the original specification. The delayed assignments to the control flow events are not problematic, i.e., are handled correctly.

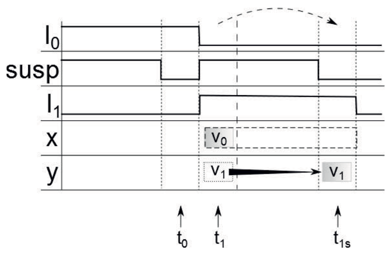

Fig. 1. Time Table for Suspend

```
  ⋮
l0: pause;
next (x) = v0;
l1: pause;
y = v1;
assert(φ(x,y));
  ⋮
```

Fig. 2. Quartz Program

3.3 Transformation for Weak Abortion

The weak preemption statements differ from their strong variants by allowing the execution of the data flow when the preemption takes place. If the abortion should take place at the termination point, it will therefore not modify the behavior. A ***weak abort***-sensitive specification should express that either the specification φ is already satisfied or the execution was aborted in a state not violating the specification, but before it was ultimately fulfilled.

Definition 9 (Transformation $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma)$). *For a given specification $\mathsf{A}\varphi$, the transformation $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma)$ is defined as*

$$\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma) := \begin{cases} \varphi, & \text{if } \varphi \text{ is propositional} \\ \sigma \vee \boldsymbol{X}\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma), & \text{if } \varphi = \boldsymbol{X}\psi \\ [\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma) \otimes (\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\gamma,\sigma) \vee \sigma \wedge \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma))], & \text{if } \varphi = [\psi \otimes \gamma] \text{ for } \otimes \in \{\underline{\mathsf{U}},\mathsf{U}\} \\ \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma) \otimes \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\gamma,\sigma), & \text{if } \varphi = \psi \otimes \gamma \text{ and } \otimes \in \{\wedge,\vee\} \end{cases}$$

The crucial point of the definition is that the specification must not be violated in a step where a weak abortion takes place. Hence, for the evaluation of a variable the value of σ is unimportant and only influences reads to the variable in a later step. This requires a different treatment of the until operators: Their evaluation must stop in a step where σ is satisfied. Furthermore, in such a step also one side of the operator must be satisfied. Hence, $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma) \vee \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\gamma,\sigma)$ must hold, but the right-hand side of this disjunction is already covered by

$$\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\gamma,\sigma) \vee \sigma \wedge (\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma) \vee \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\gamma,\sigma)) = \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\gamma,\sigma) \vee \sigma \wedge \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma)$$

and so it is enough to additionally demand $\sigma \wedge \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\psi,\sigma)$ to successfully stop the evaluation of the operator. For the next operator, the specification $\varphi = \boldsymbol{X}\psi$ must lead to a specification that requires that the execution is aborted in the first step (without restrictions) or ψ holds in the next step (with/without abortion).

Regarding the examples, we have $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathsf{G}\varphi,\sigma) = [\varphi\ \mathsf{U}\ (\sigma \wedge \varphi)]$ and $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathsf{F}\varphi,\sigma) = \mathsf{F}(\sigma \vee \varphi)$ for a propositional φ .

Correctness. The following theorem ensures the correctness of the modular proof rule for weak abortion.

Theorem 3. *For any set of guarded actions $\mathcal{G}$ and any condition σ, the following holds: $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathcal{G},\sigma) \models \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma) \leftrightarrow \mathcal{G} \models \varphi$.*

The proof is similar to the proof of Theorem 1: the used Lemma 2 is analogous for the weak abortion case, but Lemma 3 must be replaced by the following lemma:

Lemma 4. *Let $\mathcal{T}$ be the original transition system for $\mathcal{G}$ that satisfis φ and $\mathcal{T}'$ be the transition system for $\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathcal{G},\sigma)$. Assume we have paths $\pi \in \mathcal{T}$ and $\pi' \in \mathcal{T}'$ that is equivalent to π up to a minimal position where σ holds. For an arbitrary position m such that $\pi'^{(m)} \vdash \sigma$ holds, we have $(\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathcal{G},\sigma),\pi',m) \models \Theta_{\mathsf{ab}}^{\mathsf{wk}}(\varphi,\sigma)$.*

Proof. The proof can be made by an induction over the structure of φ and the fact $(\Theta_{\mathsf{ab}}^{\mathsf{wk}}(\mathcal{G},\sigma),\pi',m) \models \varphi$ which follows from the definition of $\Theta_{\mathsf{ab}}^{\mathsf{wk}}$.

With this lemma and the fact inferred from Definition $\Theta_{\mathsf{ab}}^{\mathsf{wk}}$ that the considered paths π and π' are equivalent up to t_σ, the proof is analogous to Theorem 1.

3.4 Transformation for Weak Suspension

A weak suspension freezes the control flow, but the data flow is not affected. Hence, the ***weak suspend***-sensitive specification must express that in case of a suspension, the current state is not left which motivates the following definition.

Definition 10 (Transformation $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\varphi,\sigma)$). *Given* $\Omega := \mathsf{G}(\sigma \wedge \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\gamma,\sigma))$ *and* $\otimes \in \{\wedge, \vee, \mathsf{U}\}$*, then we define*

$$\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\varphi,\sigma) := \begin{cases} [(\sigma \wedge \varphi)\ \mathsf{U}\ (\neg\sigma \wedge \varphi)], & \textit{if } \varphi \textit{ is propositional} \\ \left[\sigma\ \mathsf{U}\ \neg\sigma \wedge \boldsymbol{X}\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\psi,\sigma)\right], & \textit{if } \varphi = \boldsymbol{X}\psi \\ \left[\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\psi,\sigma)\ \underline{\mathsf{U}}\ (\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\gamma,\sigma) \vee \Omega)\right], & \textit{if } \varphi = [\psi\ \underline{\mathsf{U}}\ \gamma] \\ \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\psi,\sigma) \otimes \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\gamma,\sigma), & \textit{if } \varphi = \psi \otimes \gamma. \end{cases}$$

Regarding the examples, we have $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\mathsf{G}\varphi,\sigma) = \mathsf{G}\,[(\sigma \wedge \varphi\ \mathsf{U}\ (\neg\sigma \wedge \varphi)])$ and that $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\mathsf{F}\varphi,\sigma) = \mathsf{F}\,[(\sigma \wedge \varphi)\ \mathsf{U}\ (\neg\sigma \wedge \varphi \vee \mathsf{G}\sigma)]$ holds for a propositional φ.

It is again provable that the weak abortion is equivalent to an infinite weak suspension. The only difference to the strong case is that the weak until operator in $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\varphi,\sigma)$ is not changed, because both sides already cover the changes made in $\Theta^{\mathsf{wk}}_{\mathsf{ab}}(\varphi,\sigma)$. The term $\left[\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\psi,\sigma)\ \mathsf{U}\ \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\gamma,\sigma) \vee \mathsf{G}(\sigma \wedge (\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\psi,\sigma) \vee \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\gamma,\sigma)))\right]$ is reducible to $\left[\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\psi,\sigma)\ \mathsf{U}\ \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\gamma,\sigma)\right]$.

Correctness. The following theorem ensures the correctness of the modular proof rule for weak suspension.

Theorem 4. *For any set of guarded actions $\mathcal{G}$ and any condition σ, the following holds:* $\Theta^{\mathsf{wk}}_{\mathsf{sp}}(\mathcal{G},\sigma) \models \Theta^{\mathsf{wk}}_{\mathsf{sp}}(\varphi,\sigma) \leftrightarrow \mathcal{G} \models \varphi$.

Proof. The proof for the **weak suspend** case is analogous to the proof of Theorem 2, but the exclusion of delayed assignments to event variables (checked by $\mathsf{DFNxtEvtFree}(\mathcal{G})$) is not necessary, because all data flow assignments are executed in case of a weak suspension. Hence, the assignments to $\boldsymbol{y}$ and $\boldsymbol{x}$'s take place at t_1 and $\varphi(\boldsymbol{x},\boldsymbol{y})$ can be evaluated there, too. We illustrate this situation in Figure 3 in analogy to Figure 1. Nevertheless, a set of guarded actions containing next assignments to event variables may only satisfy $\varphi(\boldsymbol{x},\boldsymbol{y})$ during suspension, since the assignment to $\boldsymbol{x}$ is lost after t_1.

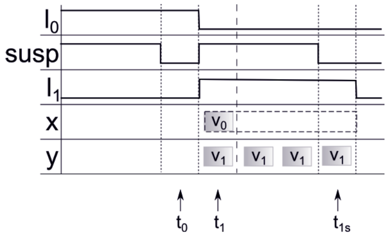

Fig. 3. Time Table for Weak Suspend

4 Example

In this section, we show how to apply the developed proof rules. To this end, let us assume that we have an already implemented traffic light controller, like the one represented by the (simplified) set of guarded actions in Figure 4 obtained from the Quartz file in Figure 5. Note that during compilation a *boot flag* `bf` is added that is ***false*** in the first macro step and ***true*** in all other steps. This is necessary for initialization purposes. The traffic light controller has one input

```
control flow:
  True ⇒ next(bf)
  ¬req∧(¬bf∨l0) ⇒ next(l0)
  req∧(¬bf∨l0) ⇒ next(l1)
  l1 ⇒ next(l2)
data flow:
  req∧(¬bf∨l0∨l2) ⇒ ylw
  l1 ⇒ grn = True
specifications:
  A G (req→grn∨ylw∧(X grn))
```

Fig. 4. Compiled Guarded Actions

```
module TrafficLightController
  (event ?req, !ylw, !grn){
  loop{
    while (¬req){
      l0: pause;
    }
    emit (ylw);
    l1: pause;
    emit (grn);
    l2: pause;
  }
} satisfies {
 A G (req→grn∨ylw∧X grn);
}
```

Fig. 5. Quartz Source Code

variable *req* and two output variables *ylw* and *grn* (indicated by *?* and *!* respectively), which are Boolean events. Thus, the outputs are ***false*** for macro steps not assigning a value to them. A traffic light usually has three lights, we will model these lights with the two output variables: *ylw=**true*** means that the yellow light is on, *grn=**true*** means that the green light is on, and *grn=**false*** means that the red light is on. The behavior of the controller is very simple, as long as the environment does not request a green light by *req=**true***, the controller will respond by not setting any output (hence, the red light is on). A request is answered by enabling the yellow light (and the red light, since *grn=**false***) in the current step, and the green light in the next step. Furthermore, it is easily provable that the controller implements the specification ***A G*** *(req → grn ∨ ylw ∧ **X** grn)*.

Assume, we want to extend the traffic light controller to operate additionally lights for a crossing pedestrian (with priority). To this end, we reuse the already existing controller, like it is done in Figure 7[2]. In Figure 6, we added the guarded actions for the compiled version where we simplified the Quartz compiler's output and for a better readability, we replaced the term *(C.l0 ∨ C.l1 ∨ C.l2)* by *inC* and *(P.l0 ∨ P.l1 ∨ P.l2)* by *inP*. The original module was used twice, but embedded in two different ***abort*** statements (in the second call the output for the yellow light is ignored, which is indicated by the underscore). It is not obvious that this implementation is correct, but we will see that our rules help to determine this.

[2] We omitted the ***immediate*** modifier for both ***abort*** statements to be consistent with the defined rules.

```
control flow:
  True ⇒ next(bf)
  ¬reqP∧¬reqC∧(bf∨(C.l2∨C.l0)∨inP)
    ⇒ next(C.l0)
  ¬reqP∧reqC∧(bf∨(C.l2∨C.l0)∨inP)
    ⇒ next(C.l1)
  ¬reqP∧C.l1∧¬reqP ⇒ next(C.l2)
  reqP∧(bf∨inC∨(P.l0∨P.l2))
    ⇒ next(P.l1)
  reqP∧P.l1 ⇒ next(P.l2)
data flow:
  bf∧reqC∧¬reqP ⇒ ylwC
  ¬reqP∧reqC∧C.l0 ⇒ ylwC
  C.l1∧¬reqP ⇒ grnC
  reqC∧C.l2∧¬reqP ⇒ ylwC
  P.l1 ⇒ grnP
  reqP∧P.l2 ⇒P.ylw
  reqP∧(C.l0∨C.l1∨C.l2) ⇒P.ylw
  reqC∧¬reqP∧(P.l0∨P.l1∨P.l2) ⇒ ylwC
```

Fig. 6. Compiled Guarded Actions

```
module TrafficLightController2
  (event ?reqC, !ylwC, !grnC,
       ?reqP, !grnP,){
  loop{
    abort{
      C: TrafficLightController
         (reqC, ylwC, grnC);
    }when (reqP);
    weak abort{
      P: TrafficLightController
          (reqP, _, grnP);
    }when(¬reqP);
  }
}
```

Fig. 7. Quartz Source Code

Applying our rules for the two ***abort*** statements after renaming the variables to the specification $\varphi(\mathit{req,ylw,grn}) = \mathbf{G}(\mathit{req} \to \mathit{grn} \vee \mathit{ylw} \wedge \mathbf{X}\ \mathit{grn})$ leads to $\Theta^{\mathsf{st}}_{\mathsf{ab}}(\varphi(\mathit{reqC,ylwC,grnC}), \mathit{reqP})$. Hence, we have to evaluate

$$\Theta^{\mathsf{st}}_{\mathsf{ab}}([(\neg \mathit{reqC} \vee \mathit{grnC} \vee \mathit{ylwC} \wedge \mathbf{X}\ \mathit{grnC})\ \mathsf{U}\ \mathsf{false}], \mathit{reqP}) =$$
$$\mathbf{G}\ (\mathit{reqC} \to \mathit{reqP} \vee \mathit{grnC} \vee \mathit{ylwC} \wedge \mathbf{X}\ (\mathit{grnC} \vee \mathit{reqP}))$$

Using the same steps we deduce the specification for the ***weak abort*** as $\Theta^{\mathsf{wk}}_{\mathsf{ab}}(\varphi(\mathit{reqP,\_,grnP}), \sigma) = [\mathit{reqP} \to \mathit{grnP} \vee \mathbf{X}\ \mathit{grnP}\ \mathsf{U}\ \neg \mathit{reqP}]$.

Hence, we know that the module calls of the *TrafficLightController* together with the surrounding ***abort*** statement satisfies the corresponding specification (without having to verify it).

Additionally, the first specification tells us that we reached the goal prioritizing the pedestrian's lights, because *reqP* is able to shadow a green light for the cars. The second specification shows that in every step, we are inside the second ***abort*** either $\mathit{reqP} \to \mathit{grnP} \vee \mathbf{X}\ \mathit{grnP}$ or $\neg \mathit{reqP}$ holds. Additionally, we know that the statement before the second ***abort*** terminates if and only if *reqP* holds. Hence, in the first step of the second ***abort*** statement, the property $\mathit{grnP} \vee \mathbf{X}\ \mathit{grnP}$ must hold. Hence, the reuse of the traffic-light controller lead to a correct implementation.

Nevertheless, we have to define similar rules for the other Quartz statements, e.g. a rule for sequences, to determine a property that is valid for the whole *TrafficLightController2* module.

5 Conclusion

In this paper, we defined transformations to modify given verification results such that these will take care of preemptions of the system. These transformations allow us to define modular proof rules for preemption statements to reason about their correctness. We are thereby able to introduce preemption statements even though these have not been considered in the available verification results, and our transformations automatically derive new specifications that hold under the preemption contexts.

References

1. André, C.: SyncCharts: A visual representation of reactive behaviors. Research Report tr95-52, University of Nice, Sophia Antipolis, France (1995)
2. Armoni, R., Bustan, D., Kupferman, O., Vardi, M.Y.: Resets vs. Aborts in linear temporal logic. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 65–80. Springer, Heidelberg (2003)
3. Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., de Simone, R.: The synchronous languages twelve years later. Proceedings of the IEEE 91(1), 64–83 (2003)
4. Berry, G., Gonthier, G.: The Esterel synchronous programming language: Design, semantics, implementation. Science of Computer Programming 19(2), 87–152 (1992)
5. Brandt, J., Schneider, K.: Separate compilation for synchronous programs. In: Falk, H. (ed.) Software and Compilers for Embedded Systems (SCOPES), Nice, France. ACM International Conference Proceeding Series, vol. 320, pp. 1–10. ACM (2009)
6. de Boer, F.S., de Roever, W.-P.: Compositional proof methods for concurrency: A semantic approach. In: de Roever, W.-P., Langmaack, H., Pnueli, A. (eds.) COMPOS 1997. LNCS, vol. 1536, pp. 632–646. Springer, Heidelberg (1998)
7. de Roever, W.-P.: The need for compositional proof systems: A survey. In: de Roever, W.-P., Langmaack, H., Pnueli, A. (eds.) COMPOS 1997. LNCS, vol. 1536, pp. 1–22. Springer, Heidelberg (1998)
8. Gesell, M., Schneider, K.: Modular verification of synchronous programs. In: Application of Concurrency to System Design (ACSD), Barcelona, Spain. IEEE Computer Society (2013)
9. Halbwachs, N.: Synchronous programming of reactive systems. Kluwer (1993)
10. Halbwachs, N.: A synchronous language at work: the story of Lustre. In: Formal Methods and Models for Codesign (MEMOCODE), Verona, Italy, pp. 3–11. IEEE Computer Society (2005)
11. Harel, D., Pnueli, A.: On the development of reactive systems. In: Apt, K. (ed.) Logic and Models of Concurrent Systems, pp. 477–498. Springer (1985)
12. Kupferman, O., Vardi, M.Y.: On the complexity of branching modular model checking (extended abstract). In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 408–422. Springer, Heidelberg (1995)
13. Le Guernic, P., Gauthier, T., Le Borgne, M., Le Maire, C.: Programming real-time applications with SIGNAL. Proceedings of the IEEE 79(9), 1321–1336 (1991)
14. Schneider, K.: Verification of Reactive Systems – Formal Methods and Algorithms. Texts in Theoretical Computer Science (EATCS Series). Springer (2003)
15. Schneider, K.: The synchronous programming language Quartz. Internal Report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany (December 2009)