Soundness Proof for a Hoare Logic for Energy Consumption Analysis*

Technical Report ICIS–R13009
Radboud University Nijmegen
Revised version - February 26, 2014

Paolo Parisen Toldin[1], Rody Kersten[2], Bernard van Gastel[3], and Marko van Eekelen[2,3]

[1] University of Bologna, Department Computer Science and Engineering, Italy
parisent@cs.unibo.it
[2] Radboud University Nijmegen, Institute for Computing and Information Sciences, The Netherlands
{r.kersten,m.vaneekelen}@cs.ru.nl
[3] Open University of the Netherlands, Faculty of Management, Science and Technology, The Netherlands
{bernard.vangastel,marko.vaneekelen}@ou.nl

Abstract. Energy inefficient software implementations may cause battery drain for small systems and high energy costs for large systems. Dynamic energy analysis is often applied to mitigate these issues. However, this is often hardware-specific and requires repetitive measurements using special equipment. We present a static analysis deriving upper-bounds for energy consumption based on an introduced energy-aware Hoare logic. Software is considered together with models of the hardware it controls. The Hoare logic is parametric with respect to the hardware. Energy models of hardware components can be specified separately from the logic. Parametrised with one or more of such component models, the analysis can statically produce a sound (over-approximated) upper-bound for the energy-usage of the hardware controlled by the software.

1 Introduction

Power consumption and green computing are nowadays important topics in IT. From small systems such as wireless sensor nodes, cell-phones and embedded devices to big architectures such as data centers, mainframes and servers, energy consumption is an important factor. Small devices are often powered by a battery, which should last as long as possible. For larger devices, the problem lies mostly with the costs of powering the device. These costs are often amplified by inefficient power-supplies and cooling of the system.

Obviously, power consumption depends not only on hardware, but also on the software *controlling* the hardware. Currently, most of the methods available to programmers to analyse energy consumption caused by software use dynamic analysis: measuring while the software is running. Measuring power consumption of a system and especially of its individual components is not a trivial task. A designated measuring set-up is required. This means that most programmers currently have no idea how much energy their software consumes. A static analysis of energy consumption would be a big improvement, potentially leading to more energy-efficient software. Such a static analysis is presented in this paper.

Since the software interacts with multiple components (software and hardware), energy consumption analysis needs to incorporate different kinds of analysis. Power consumption may depend on hardware state, values of variables and bounds on the required number of clock-cycles.

Related Work There is a large body of work on energy-efficiency of software. Most papers approach the problem on a high level, defining programming and design patterns for writing energy-efficient code e.g. [1–3]. In [4], a modular design for energy-aware software is presented that is based on a series of rules on UML schemes. The language itself and its programming style are considered in [5, 6], where the program is divided into "phases" describing similar behaviour. Based on the behaviour of the software, design level optimisations are proposed to achieve lower energy consumption. A lot of research is dedicated to building compilers that

* This work is partially financed by the IOP GenCom GoGreen project, sponsored by the Dutch Ministry of Economic Affairs, Agriculture and Innovation.

optimize code for energy-efficiency [7–9]. In [10] and [11], energy modeling techniques are proposed based on Petri-nets. The processor is the only component that is modeled. Analyses for consumption of generic resources are discussed in [12–15]. The main differences with our work are that we include an explicit hardware model and a context in the form of component states. This context enables the inclusion of state-dependent energy consumption. Relatively close to our approach are [16] and [17], in which energy consumption of embedded software is analysed. The main difference with our work is that these are aimed at specific architectures, whereas our approach is hardware parametric with respect to the hardware.

Our Approach Contrary to the approaches above, we are interested in statically deriving bounds on energy-consumption using a novel, generic approach that is parametrised with hardware models. Energy consumption analysis is an instance of resource consumption analysis. Other instances are worst-case execution time [18], size [19], loop bound [20, 21] and memory [21] analysis). The focus of this paper is on energy analysis. Energy consumption models of hardware components are input for our analysis. The analysis requires information about the software, such as dependencies between variables and information about the number of loop iterations. For this reason we assume that a previous analysis (properly instantiated for our case) has been made deriving loop bounds (e.g. [22, 21]) and variable dependency information (e.g. [23]).

Our approach is essentially an energy-aware Hoare logic that is proven sound with respect to an energy-aware semantics. Both the semantics and the logic assume energy-aware component models to be present. The central control is however assumed to be in the software. Consequently, the analysis is done on a hybrid system of software and models of hardware components. The Hoare logic yields an upper bound on the energy consumption of a system of hardware components that are controlled by software.

Our contribution The main contributions of this paper are:

- A novel energy-aware software semantics that is parametrised with hardware models.
- A corresponding energy-aware Hoare logic that enables formal reasoning about energy consumption such as deriving an upper-bound for the energy consumption of the system.
- An example of the use of the logic comparing two programs that control a wireless sensor node.
- A soundness proof of the derived upper-bounds with respect to the semantics.

The basic modelling and semantics are presented in Sect. 2. Energy-awareness is added and the logic is presented in Sect. 3. An example is given in Sect. 4 and the soundness is proved in Sect. 5. The paper is concluded in Sect. 6.

2 Modelling Hybrid Systems

Most modern electronic systems consist of hardware and software. In order to study the energy consumption of such hybrid systems we will consider both hardware and software in one single modelling framework. This section defines a hybrid logic in which software plays a central role controlling hardware components. The hardware components are modelled in such a way that only the relevant information for the energy consumption analysis is present. In this paper, the controlling software is assumed to be written in a small language designed just for illustrating the analysis.

2.1 Language

Our analysis is performed on a 'while' language. As we just use this language for illustration purposes, the only supported type in the language is unsigned integer. There are no explicit booleans. The value 0 is handled as a **False** value, while all the other possible values are handled as a **True** value. There are no global variables and parameters are passed by-value, so functions do not have side-effects on the program state. Recursion is not supported, and functions are statically scoped. However, there is support for `while` loops. There are explicit statements for operations on hardware components, like the processor, memory, storage or network devices. By explicitly introducing these statements it is easier to reason about those components, as opposed to, for instance, using conventions about certain memory regions that will map to certain hardware devices. Functions on components can have a fixed number of arguments and always return a value. To refer to a particular function f of a component C_i we use the notation $C_i :: f$.

The grammar for our language is defined as follows:

$$
\begin{aligned}
c \in \textsc{Const} &= n \in \mathbb{N} \\
x \in \textsc{Var} &= \text{'A'} \mid \text{'B'} \mid \text{'C'} \mid \ldots \\
e \in \textsc{Expr} &= c \mid x \mid x = e_1 \mid e_1 \boxdot e_2 \mid C_i :: f(e_1) \mid f(e_1) \mid S; e_1 \\
S \in \textsc{Statement} &= \textbf{skip} \mid S_1; S_2 \mid e \mid \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if} \\
&\quad \mid \textbf{while } e \textbf{ do } S \textbf{ end while} \mid F\ S_1 \\
F \in \textsc{Func} &= \textbf{function } f(x) \textbf{ begin } e \textbf{ end}
\end{aligned}
$$

Where $\boxdot \in \{+, -, *, >, \geq, \equiv, \not\equiv, \leq, <, \wedge, \vee\}$.

2.2 Modelling Components

To reason about hybrid systems we need a way to model hardware components (e.g. memory, harddisk, network controller) that captures the behaviour of those components with respect to resource consumption. Hence, we introduce a *component model* that consists of a state and a set of functions that can change the state: *component functions*. A component state $C_i :: s$ is a collection of variables of any type. They can signify e.g. that the component is on, off or in stand-by.

A component function is modelled by a function that produces the return value (rv_f) and a function that updates the internal state of the component (δ_f). Both functions are functions over the state variables. The update function $C_i :: \delta_f$ and the return value function $C_i :: rv_f$ take the state s and the arguments *args* passed to the component function and return respectively the new state of the component and the return value. Each component C_i may have multiple component functions. All the state changes in components must be explicit in the source code as an operation, a *component function*, on that specific component.

2.3 Semantics

Standard, non-energy-aware semantics can be defined for our language. Full semantics are given in a technical report [24]. Below, the assignment rule (*sAssign*) and the component function call rule (*sCallCmpF*) are given to illustrate the notation and the way of handling components.

$$
\frac{\Delta \vdash \langle e, \sigma, \Gamma \rangle \Downarrow^e \langle n, \sigma', \Gamma' \rangle}{\Delta \vdash \langle x_1 = e, \sigma, \Gamma \rangle \Downarrow^e \langle n, \sigma'[x_1 \leftarrow n], \Gamma' \rangle} (\text{sAssign})
$$

$$
\frac{\Delta \vdash \langle e, \sigma, \Gamma \rangle \Downarrow^e \langle a, \sigma', \Gamma' \rangle \quad C_i :: rv_f(C_i^\Gamma :: s, a) = n \quad \Gamma' = \Gamma[C_i :: s \leftarrow C_i :: \delta_f(C_i^\Gamma :: s, a)]}{\Delta \vdash \langle C_i :: f(e), \sigma, \Gamma \rangle \Downarrow^e \langle n, \sigma, \Gamma' \rangle} (\text{sCallCmpF})
$$

The rules are defined over a triple $\langle e, \sigma, \Gamma \rangle$ with respectively a program expression e (or statement S), the program state function σ and the component state environment Γ. The *program* state function returns for every variable its actual value. Δ is an environment of function definitions. We use the following notation for substitution: $\sigma[x_i \leftarrow n]$. The reduction symbol $\Downarrow^e$ is used for expressions, which evaluate to a value and a new state function. We use $\Downarrow^s$ for statements, which only evaluate to a new state function.

In the following sections we will define energy-aware semantics and energy analysis rules. We used a consistent naming scheme for the different variants of the rules (e.g. **s**Assign, **e**Assign and **a**Assign for the Assignment rule in respectively the **s**tandard non-energy-aware semantics, the **e**nergy aware semantics and the energy **a**nalysis rules). The full semantics are given in Fig. **??**.

3 Energy Analysis of Hybrid Systems

In this section we extend our hybrid logic in order to reason about the energy consumption of programs. We distinguish two kinds of energy usage: *incidental* and *time-dependent*. The former represents an operation that uses a constant amount of energy, disregarding any time aspect. The latter signifies a change in the state of the component; while a component is in a certain state it is assumed to draw a constant amount of energy *per time unit*.

$$\frac{}{\Delta \vdash \langle c, \sigma, \Gamma\rangle \Downarrow^e \langle c, \sigma, \Gamma\rangle}(\text{sConst}) \quad \frac{}{\Delta \vdash \langle x, \sigma, \Gamma\rangle \Downarrow^e \langle \sigma(x), \sigma, \Gamma\rangle}(\text{sVar})$$

$$\frac{\Delta \vdash \langle e_1, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle \quad \Delta \vdash \langle e_2, \sigma', \Gamma'\rangle \Downarrow^e \langle m, \sigma'', \Gamma''\rangle \quad C_{imp} :: \boxdot(n, m) = p}{\Delta \vdash \langle e_1 \boxdot e_2, \sigma, \Gamma\rangle \Downarrow^e \langle p, \sigma'', \Gamma''\rangle}(\text{sBinOp})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle}{\Delta \vdash \langle x = e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma'[x \leftarrow n], \Gamma'\rangle}(\text{sAssign})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle a, \sigma', \Gamma'\rangle \quad C_i :: rv_f(C_i^\Gamma :: s, a) = n \quad \Gamma' = \Gamma[C_i :: s \leftarrow C_i :: \delta_f(C_i^\Gamma :: s, a)]}{\Delta \vdash \langle C_i :: f(e), \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma, \Gamma'\rangle}(\text{sCallCmpF})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle a, \sigma', \Gamma'\rangle \quad \Delta(f) = (e_1, \Delta', x) \quad \Delta' \vdash \langle e_1, [x \leftarrow a], \Gamma'\rangle \Downarrow^e \langle n, \sigma'', \Gamma''\rangle}{\Delta \vdash \langle f(e), \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma''\rangle}(\text{sCallF})$$

$$\frac{\Delta \vdash \langle S, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle \quad \Delta \vdash \langle e, \sigma', \Gamma'\rangle \Downarrow^e \langle n, \sigma'', \Gamma''\rangle}{\Delta \vdash \langle S, e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma'', \Gamma''\rangle}(\text{sExprConcat})$$

$$\frac{\Delta \vdash \langle e_1, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle}{\Delta \vdash \langle e_1, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle}(\text{sExprAsStmt}) \quad \frac{}{\Delta \vdash \langle \textbf{skip}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma, \Gamma\rangle}(\text{sSkip})$$

$$\frac{\Delta \vdash \langle S_1, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle \quad \Delta \vdash \langle S_2, \sigma', \Gamma'\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}{\Delta \vdash \langle S_1; S_2, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}(\text{sStmtConcat})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle 0, \sigma', \Gamma'\rangle \quad \Delta \vdash \langle S_2, \sigma', \Gamma'\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}{\Delta \vdash \langle \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}(\text{sIf-False})$$

$$\frac{n \neq 0 \quad \Delta \vdash \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle \quad \Delta \vdash \langle S_1, \sigma', \Gamma'\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}{\Delta \vdash \langle \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}(\text{sIf-True})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle 0, \sigma', \Gamma'\rangle}{\Delta \vdash \langle \textbf{while } e \textbf{ do } S_1 \textbf{ end while}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle}(\text{sWhile-False})$$

$$\frac{n \neq 0 \quad \Delta \vdash \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle \quad \Delta \vdash \langle S_1; \textbf{while } e \textbf{ do } S_1 \textbf{ end while}, \sigma', \Gamma'\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}{\Delta \vdash \langle \textbf{while } e \textbf{ do } S_1 \textbf{ end while}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}(\text{sWhile-True})$$

$$\frac{\Delta[f \leftarrow (e, \Delta, x)] \vdash \langle S, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle}{\Delta \vdash \langle \textbf{function } f(x) \textbf{ begin } e \textbf{ end } S, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle}(\text{sFuncDef})$$

Fig. 1. Energy-aware semantics.

3.1 Energy-Aware Semantics

As energy consumption can be based on time, we first need to extend our semantics to be time-aware. We effectively extend all the rules of the semantics with an extra argument, a global timestamp $\mathfrak{t}$. Using this timestamp we are able to model and analyse time-dependent energy usage.

We track energy usage for each component individually, by using an accumulator $\mathfrak{e}$ that is added to the component model. For time-dependent energy usage, with each component state change, the energy used while the component was in the previous state is added to the accumulator. To enable calculation of the time spent in the current state, we add τ to the component model, signifying the timestamp at which the component entered the current state. We assume that each component has a constant *power draw* while in a state. Therefore, the component model function $C_i :: \phi(s)$ maps component states onto the corresponding power draw, independent of time. To calculate the power consumed while in a certain state we define the *td* function, with as arguments the component and the current timestamp:

$$td(C_i, t) = C_i :: \phi(s) \cdot (t - C_i :: \tau)$$

We model *incidental energy usage* associated with a component function f with the constant $C_i :: \mathfrak{E}_f$. For each call to a component function we add this constant to the energy accumulator.

A component function call can influence energy consumption in two ways: through its associated incidental energy consumption and by changing the state, thereby influencing time-dependent energy usage. This is expressed by the energy-aware semantic rule ($eCallCmpF$) for component functions as defined below, with $C_i :: \mathfrak{T}_f$ representing the time it costs to execute this component function.

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle a, \sigma', \Gamma', \mathfrak{t}'\rangle \quad C_i :: rv_f(C_i^{\Gamma'} :: s, a) = n \quad \Gamma'' = \Gamma[C_i :: \mathfrak{e} \mathrel{+}= C_i :: \mathfrak{E}_f + td(C_i^{\Gamma'}, \mathfrak{t}), C_i :: s \leftarrow C_i :: \delta_f(C_i^{\Gamma'} :: s, a), C_i :: \tau \leftarrow \mathfrak{t}']}{\Delta \vdash \langle C_i :: f(e), \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma, \Gamma'', \mathfrak{t}' + C_i :: \mathfrak{T}_f\rangle}(\text{eCallCmpF})$$

Note the addition of the incidental and time dependent energy usages ($C_i :: \mathfrak{E}_f$ and $td(C_i^\Gamma, \mathfrak{t})$ respectively) to the energy accumulator $C_i :: \mathfrak{e}$, the incrementation of the global time with $C_i :: \mathfrak{T}_f$ and the update of the

component timestamp $C_i :: \tau$. Evaluation by $\Downarrow$ in the energy-aware semantics extends the original semantics with a timestamp and an energy accumulator, which are used to calculate the total energy consumption of the evaluation ($\mathfrak{e}_{system}$ as defined below). The full energy-aware semantics are given in Fig. 2.

The energy accumulator of the components is not always up to date with respect to the current time, as it is only updated in the (*eCallCmpF*) rule. This is done for simplicity; otherwise each rule that adjusts the global time needs to update the energy accumulator of all components.

To calculate the total actual energy usage, the time the components are in their current state should still be accounted for. This means we have to add the result of the *td* function for each component. The total energy consumption of the system can be calculated at any time as follows:

$$\mathfrak{e}_{system}(\Gamma, \mathfrak{t}) = \sum_i C_i^\Gamma :: \mathfrak{e} + td(C_i^\Gamma, \mathfrak{t})$$

We can now make the distinction between non-energy-aware component state $C_i :: s$, and energy-aware component state, which also includes the time-stamp τ and the energy accumulator $\mathfrak{e}$.

Most energy consuming actions are explicit in our language: $C_i :: consume()$. However, basic language features, such as evaluation of arithmetic expressions, also implicitly consume energy. We capture this behaviour in the C_{imp} component. This component is an integral part of our energy-aware semantics and logic. The C_{imp} component should at least have resource consumption constants defined for the following operations:

- $C_{imp} :: \mathfrak{E}_e$ and $C_{imp} :: \mathfrak{T}_e$ for expression evaluation.
- $C_{imp} :: \mathfrak{E}_a$ and $C_{imp} :: \mathfrak{T}_a$ for assignment.
- $C_{imp} :: \mathfrak{E}_w$ and $C_{imp} :: \mathfrak{T}_w$ for while loops.
- $C_{imp} :: \mathfrak{E}_{ite}$ and $C_{imp} :: \mathfrak{T}_{ite}$ for conditionals.

To capture the resource consumption of these basic operations, we extend the associated rules in the semantics. The energy-aware rule for assignment (*eAssign*) is listed below, with $C_{imp} :: \mathfrak{E}_a$ for the incidental energy usage of an assignment and $C_{imp} :: \mathfrak{T}_a$ for the time it takes to perform an assignment.

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}' \rangle \quad \Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\Delta \vdash \langle x = e, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma'[x \leftarrow n], \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_a \rangle} (\text{eAssign})$$

All computations of resource consumption and new component states are done symbolically. In the logic, these values are added, multiplied and subtracted or their `max` is taken. Hence, every $\mathfrak{t}$, $\mathfrak{e}$ and τ, as well as the values in component states, are polynomial expressions, extended with the `max` operator, over program variables. Additionally, symbolic states are used, both as input for the program and as start state for the components. The aforementioned polynomials also range over the symbols used in these symbolic states.

3.2 Energy Aware Modelling

Energy-aware models will be used to derive upper-bounds for energy consumption of the modelled system. In order for the energy-aware model to be suited for the analysis the model should reflect an upper-bound on the actual consumption. This can be based on detailed documentation or on actual energy measurements.

To provide a sound analysis, we need to assume that components are modelled in such a way that the component states reflect different power-levels and are partially ordered. Greater states should imply greater power draw. We will use finite state models only to enable fixpoint calculation in our analysis of while loops.

Component states should be partially ordered to enable calculation of an overestimation. A component should thus have finitely many states. Therefore, every variable should have an associated finite domain. It comes easy then to define a partial order with a least upper bound, a lattice.

Definition 1 (Partial order of component states). *For a component C_i, $C_i :: s_1 \geq C_i :: s_2$ iff for every variable v in the component state $C_i :: s_1.v \geq C_i :: s_2.v$.*

Definition 2 (Least upper bound of component states). $\mathbf{lub}(C_i :: s_1, C_i :: s_2)$ *takes for every integer i in the component states $max(s_1.i, s_2.i)$.*

Bigger states cannot consume less energy than smaller states. In other words, power draw functions ϕ preserve the ordering.

$$\frac{}{\Delta \vdash \langle c, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle c, \sigma, \Gamma, \mathfrak{t}\rangle}(\text{eConst}) \qquad \frac{}{\Delta \vdash \langle x, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle \sigma(x), \sigma, \Gamma, \mathfrak{t}\rangle}(\text{eVar})$$

$$\frac{\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \qquad C_{imp} :: \boxdot(n, m) = p \quad \Delta \vdash \langle e_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^e \langle m, \sigma'', \Gamma'', \mathfrak{t}''\rangle \quad \Gamma''' = \Gamma''[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\Delta \vdash \langle e_1 \boxdot e_2, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle p, \sigma'', \Gamma''', \mathfrak{t}'' + C_{imp} :: \mathfrak{T}_e\rangle}(\text{eBinOp})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\Delta \vdash \langle x = e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma'[x \leftarrow n], \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_a\rangle}(\text{eAssign})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle a, \sigma', \Gamma', \mathfrak{t}'\rangle \quad C_i :: rv_f(C_i^{\Gamma'} :: s, a) = n \quad \Gamma'' = \Gamma[C_i :: \mathfrak{e} \mathrel{+}= C_i :: \mathfrak{E}_f + td(C_i^{\Gamma'}, \mathfrak{t}), C_i :: s \leftarrow C_i :: \delta_f(C_i^{\Gamma'} :: s, a), C_i :: \tau \leftarrow \mathfrak{t}']}{\Delta \vdash \langle C_i :: f(e), \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma, \Gamma'', \mathfrak{t}' + C_i :: \mathfrak{T}_f\rangle}(\text{eCallCmpF})$$

$$\frac{\Delta(f) = (e_1, \Delta', x) \quad \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle a, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Delta' \vdash \langle e_1, [x \leftarrow a], \Gamma', \mathfrak{t}'\rangle \Downarrow^e \langle n, \sigma'', \Gamma'', \mathfrak{t}''\rangle}{\Delta \vdash \langle f(e), \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma'', \mathfrak{t}''\rangle}(\text{eCallF})$$

$$\frac{\Delta \vdash \langle S, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Delta \vdash \langle e, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^e \langle n, \sigma'', \Gamma'', \mathfrak{t}''\rangle}{\Delta \vdash \langle S, e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma'', \Gamma'', \mathfrak{t}''\rangle}(\text{eExprConcat})$$

$$\frac{\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle}{\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle}(\text{eExprAsStmt}) \qquad \frac{}{\Delta \vdash \langle \textbf{skip}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma, \Gamma, \mathfrak{t}\rangle}(\text{eSkip})$$

$$\frac{\Delta \vdash \langle S_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Delta \vdash \langle S_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}''\rangle}{\Delta \vdash \langle S_1; S_2, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}''\rangle}(\text{eStmtConcat})$$

$$\frac{\Delta \vdash \langle S_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}''\rangle \quad \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle 0, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Gamma''' = \Gamma''[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_{ite}]}{\Delta \vdash \langle \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' + C_{imp} :: \mathfrak{T}_{ite}\rangle}(\text{eIf-False})$$

$$\frac{n \neq 0 \qquad \Delta \vdash \langle S_1, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}''\rangle \quad \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Gamma''' = \Gamma''[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_{ite}]}{\Delta \vdash \langle \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' + C_{imp} :: \mathfrak{T}_{ite}\rangle}(\text{eIf-True})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle 0, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w]}{\Delta \vdash \langle \textbf{while } e \textbf{ do } S_1 \textbf{ end while}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w\rangle}(\text{eWhile-False})$$

$$\frac{\Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Delta \vdash \langle S_1; \textbf{while } e \textbf{ do } S_1 \textbf{ end while}, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle \quad n \neq 0}{\Delta \vdash \langle \textbf{while } e \textbf{ do } S_1 \textbf{ end while}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle}(\text{eWhile-True})$$

$$\frac{\Delta[f \leftarrow (e, \Delta, x)] \vdash \langle S, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle}{\Delta \vdash \langle \textbf{function } f(x) \textbf{ begin } e \textbf{ end } S, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle}(\text{eFuncDef})$$

Fig. 2. Energy-aware semantics.

Axiom 1 (Power draw function preserves ordering) *Let* $s_1 = C_i^{\Gamma_1} :: s$, $s_2 = C_i^{\Gamma_2} :: s$, *if* $s_1 \geq s_2$ *then* $C_i :: \phi(s_1) \geq C_i :: \phi(s_2)$.

Component state update functions δ preserve the ordering. For this reason, δ_f cannot depend on the arguments of f. To signify this, we will use $\delta(s)$ instead of $\delta(s, args)$ in the logic.

Axiom 2 (Component state update function preserves ordering) *Let* $C_i :: \delta_f(s_1) = s_1'$ *and* $C_i :: \delta_f(s_2) = s_2'$, *if* $s_1 \geq s_2$, *then* $s_1' \geq s_2'$.

Energy-aware component states are partially ordered. This ordering extends the ordering on component states in a natural way by adding an energy accumulator and a timestamp. The timestamp reflects the time a component has spent in a certain state. So, the earliest timestamp reflects the highest energy usage. Therefore, with respect to timestamps the energy-aware component state ordering should be defined such that smaller timestamps lead to bigger energy-aware component states.

Definition 3 (Ordering of Energy-Aware Component States). *Given two pair* $\Gamma_1, \mathfrak{t}_1$ *and* $\Gamma_2, \mathfrak{t}_2$, *we say that* $(\Gamma_1, \mathfrak{t}_1) \geq (\Gamma_2, \mathfrak{t}_2)$ *if* $\forall i. td(C_i^{\Gamma_1}, \mathfrak{t}_1) + C_i^{\Gamma_1} :: \mathfrak{e} \geq td(C_i^{\Gamma_2}, \mathfrak{t}_2) + C_i^{\Gamma_2} :: \mathfrak{e}$.

Definition 4 (Least upper bound of Γ). *For two sets of states of the same components,* Γ_1 *and* Γ_2, *the least upper bound is defined as follows.*

$C_i^{\mathbf{lub}(\Gamma_1, \Gamma_2)}$ *is such that*

- $C_i^{\mathbf{lub}(\Gamma_1,\Gamma_2)} :: s = \mathbf{lub}(C_i^{\Gamma_1} :: s, C_i^{\Gamma_2} :: s)$
- $C_i^{\mathbf{lub}(\Gamma_1,\Gamma_2)} :: \mathfrak{e} = \max\{C_i^{\Gamma_1} :: \mathfrak{e}, C_i^{\Gamma_2} :: \mathfrak{e}\}$
- $C_i^{\mathbf{lub}(\Gamma_1,\Gamma_2)} :: \tau = \min\{C_i^{\Gamma_1} :: \tau, C_i^{\Gamma_2} :: \tau\}$

Severeness of model restrictions There are several restrictions to the modelling that may seem far from reality.

1. *Component state functions take up a constant amount of time and incidental energy.* This is needed for the soundness proof. For instance, when a radio component sends a message, the duration of the function call cannot directly depend on the number of bytes in the message. In most cases this can be dealt with by using a different way of modelling. First, one can use an overestimation. Second, such dependencies can be removed by distributing the costs over multiple function calls. For instance, the radio component can have a function to send a fixed number of bytes. If it internally keeps a queue, the additional costs of sending the full queue can be modelled by distributing it over separate queueing operations. energy consumption of components must remain fixed per component state.
2. *With each component state a constant power draw is associated.* However, some hardware may accumulate heat over time incurring increasing energy consumption over time. Such a 'heating' problem can be modelled e.g. by changing state to a higher energy level with every call of a component function. This is still an approximation of course. In the future, we want to study models with time driven state change or with time-dependent power draw.
3. *Component model must be finite state machines.* Modelling systems with finite state machines is not uncommon, e.g using model checking and the right kind of abstraction for the property that is studied. In our models the abstraction should be such that the energy consumption is modelled as close as possible.
4. *The effect of component state functions on the component states cannot depend on the arguments of the function.* Also, component models cannot influence each other. Both restrictions are needed for soundness guarantee of our analysis. This restricts the modelling. Using multiple component state functions instead of dynamic arguments and cross-component calls is a way of modelling that can mitigate these restrictions in certain cases. Relieving these restrictions in general is part of future work.

3.3 A Hoare Logic for Energy Analysis

This section treats the definition of an energy-aware logic with energy analysis rules that can be used to bound the energy consumption of the analysed system. The full set of rules is given in Fig. 3. These rules are deterministic; at each moment only one rule can be applied.

$$\frac{}{\{\Gamma;\mathfrak{t};\rho\}n\{\Gamma;\mathfrak{t};\rho\}}(\text{aConst}) \qquad \frac{}{\{\Gamma;\mathfrak{t};\rho\}x\{\Gamma;\mathfrak{t};\rho\}}(\text{aVar})$$

$$\frac{\{\Gamma;\mathfrak{t};\rho\}e_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \quad \{\Gamma_1;\mathfrak{t}_1;\rho_1\}e_2\{\Gamma_2;\mathfrak{t}_2;\rho_2\} \quad \Gamma_3 = \Gamma_2[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma;\mathfrak{t};\rho\}e_1 \boxdot e_2\{\Gamma_3;\mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_e;\rho_2\}}(\text{aBinOp})$$

$$\frac{\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \quad \Gamma_2 = \Gamma_1[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\{\Gamma;\mathfrak{t};\rho\}x = e\{\Gamma_2;\mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_a;\rho_2\}}(\text{aAssign})$$

$$\frac{\Gamma_1 = \Gamma[C_i :: s \leftarrow C_i :: \delta_f(C_i :: s), C_i :: \tau \leftarrow \mathfrak{t}, C_i :: \mathfrak{e} \mathrel{+}= C_i :: \mathfrak{E}_f + td(C_i, \mathfrak{t})]}{\{\Gamma;\mathfrak{t};\rho\}C_i :: f(args)\{\Gamma_1;\mathfrak{t} + C_i :: \mathfrak{T}_f;\rho\}}(\text{aCallCmpF})$$

$$\frac{\begin{array}{c}\Delta(f) = (e_1, x) \qquad \{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \\ e = a \in \rho \qquad \{\Gamma_1;\mathfrak{t}_1;\rho_1[x' \leftarrow a]\}e_1[x \leftarrow x']\{\Gamma_2;\mathfrak{t}_2;\rho_2\} \quad x' \text{ fresh in } e_1\end{array}}{\{\Gamma;\mathfrak{t};\rho\}f(e)\{\Gamma_2;\mathfrak{t}_2;\rho_2\}}(\text{aCallF})$$

$$\frac{}{\{\Gamma;\mathfrak{t};\rho\}\mathbf{skip}\{\Gamma;\mathfrak{t};\rho\}}(\text{aSkip}) \qquad \frac{\{\Gamma;\mathfrak{t};\rho\}S_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \quad \{\Gamma_1;\mathfrak{t}_1;\rho_1\}S_2\{\Gamma_2;\mathfrak{t}_2;\rho_2\}}{\{\Gamma;\mathfrak{t};\rho\}S_1; S_2\{\Gamma_2;\mathfrak{t}_2;\rho_2\}}(\text{aConcat})$$

$$\frac{\begin{array}{cc}\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} & \{\Gamma_2;\mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_{ite};\rho_1\}S_1\{\Gamma_3;\mathfrak{t}_2;\rho_2\} \\ \Gamma_2 = \Gamma_1[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_{ite}] & \{\Gamma_2;\mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_{ite};\rho_1\}S_2\{\Gamma_4;\mathfrak{t}_3;\rho_3\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\mathbf{if}\ e\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{end\ if}\{\mathbf{lub}(\Gamma_3,\Gamma_4);\max\{\mathfrak{t}_2,\mathfrak{t}_3\};\rho_4\}}(\text{aIf})$$

$$\frac{\begin{array}{cc}\Gamma_1 = \mathbf{process\text{-}td}(\Gamma,\mathfrak{t}) & \Gamma_3 = \Gamma_2[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \\ \{\mathbf{wci}(\Gamma_1, e; S);\mathfrak{t};\rho\}e\{\Gamma_2;\mathfrak{t}_1;\rho_1\} & \{\Gamma_3;\mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_w;\rho_1\}S\{\Gamma_4;\mathfrak{t}_2;\rho_2\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\mathbf{while}_{ib}\ e\ \mathbf{do}\ S\ \mathbf{end\ while}\{\mathbf{oe}(\Gamma_1,\mathfrak{t},\Gamma_4,\mathfrak{t}_2,ib);\rho_3\}}(\text{aWhile})$$

Fig. 3. Energy analysis rules.

Our energy consumption analysis depends on external symbolic analysis of variables and loop analysis. The results of this external analysis are assumed to be accessible in our Hoare Logic in two ways.

Firstly, we restrict the scope of our analysis to programs that are bound in terms of execution. We assume that all loops and component functions terminate on any input. Each loop is annotated with a bound: $\mathbf{while}_{ib}$. The bound is a polynomial over the input variables, which expresses an upper-bound on the number of iterations of the loop. We have added the ib to the while rule in the energy analysis rules to make this assumption explicit. Derivation of bounds is considered out of scope for our analysis. We assume that an external analysis has produced a sound bound.

Secondly, the symbolic state environment ρ gives us a symbolic state of every variable for each line of code, e.g, $\{x_1 = e_1\}x_1 = x_1 + x_2 + x_3\{x_1 = e_1 + x_2 + x_3\}$, plus other non-energy related properties invariants that have already been proven. In Fig. 3 we included this prerequisite by explicitly denoting it as ρ, ρ_1, ...

All the judgements in the rules have the following shape: $\{\Gamma; \mathfrak{t}; \rho\}S\ \{\Gamma'; \mathfrak{t}'; \rho'\}$, where Γ is the set of all energy aware component states, $\mathfrak{t}$ is the global time and ρ represents the symbolic state environment retrieved from the earlier standard analysis. We use the notation $\Gamma[n \mathrel{+}= m]$ as a shorthand for $\Gamma[n \leftarrow n + m]$. As (energy-aware) component states are partially ordered, we can take a least upper bound of states $\mathbf{lub}(s_1, s_2)$ and sets of energy-aware component states $\mathbf{lub}(\Gamma_1, \Gamma_2)$.

We want to point the user to the most relevant aspects of the rules. The rule (*aCallCmpF*) uses the $td(C_i, t)$ function to estimate the time-dependent energy consumption of component function calls. The (*aIf*) rule takes the least upper bound of the energy-aware component states and the maximum of the time estimates.

Special attention is warranted for the (*aWhile*) rule. We study the body of the while loop in isolation. This requires processing the time-dependent energy consumption that occurred before the loop (**process-td**). An over-estimation (**oe**) of the energy consumption of the loop will be calculated by taking the product of the bound on the number of iterations and an over-estimation of the energy consumption of a single iteration, i.e. the worst-case iteration (**wci**). The worst-case-iteration is determined by taking the least upper-bound of the set of all states that can occur during the execution of the loop. As there are a finite number of states for each component, this set can be determined via a fix point construction (**fix**). The fixpoint is calculated by iterating the component iteration function (**ci**).

In order to support the analysis of statements after the loop, also an over-estimation of the component states after the loop has to be calculated. For brevity in Fig. 3, this is dealt with in the calculation of **oe**.

Five calculations are needed:

1. Component iteration function **ci**. The component iteration function $\mathbf{ci}_i(S)$ aggregates the (possibly over-estimated) effects of S on C_i. It performs the analysis on S, then considers only the effects on C_i. If there are nested loops or conditionals, the effects on the state of C_i are overestimated in the same manner as in the rest of the analysis. By $\mathbf{ci}_i^n(S)$ we mean the component iteration function applied n times: $\mathbf{ci}_i(S) \circ \mathbf{ci}_i^{n-1}(S)$, with $\mathbf{ci}_i^1(S) = \mathbf{ci}_i(S)$.
2. Fixpoint function **fix**. Because component states are finite, there is an iteration after which a component is in a state that it has already been in, earlier in the loop (unless the loop is already finished before this point is reached). Since components are independent, the behaviour of the component will be identical to its behaviour the first time it was in this state. This is a fixpoint on the set of component states that can be reached in the loop. It can be found using the $\mathbf{fix}_i(S)$ function, which finds the smallest n for which $\exists k.\mathbf{ci}_i^{n+1}(S) = \mathbf{ci}_i^k(S)$. The number of possible component states is an upper bound for n.
3. Worst-Case Iteration function **wci**. To make a sound overestimation of the energy consumption of a loop, we need to find the iteration that consumes the most. As our analysis is non-decreasing with respect to component states, this is the iteration which starts with the largest component state in the precondition. For this purpose, we introduce the worst-case iteration function $\mathbf{wci}_i(S)$, which computes the least-upper bound of all the states up to the fixpoint: $\mathbf{wci}_i(S) = \mathbf{lub}(\mathbf{ci}_i^0(S), \mathbf{ci}_i^1(S), \ldots, \mathbf{ci}_i^{\mathbf{fix}_i(S)}(S))$. The global version $\mathbf{wci}(\Gamma, S)$ is defined by iteratively applying the $\mathbf{wci}_i(S)$ function to each component C_i in Γ.
4. Overestimation function **oe**. This function overestimates the energy-aware output states of the loop. It needs to do three things: find the largest non-energy-aware output states, find the minimal timestamps and add the resource consumption of the loop itself. This function gets as input: the start state of the loop Γ_{in}, the start time $\mathfrak{t}$, the output state from the analysis of the worst-case iteration Γ_{out}, the end time from the analysis of the worst-case iteration $\mathfrak{t}'$ and the iteration bound ib. It returns an overestimated energy-aware component state and an overestimated global time.

Because component state update functions δ preserve the ordering, the analysis of the worst-case iteration results in the maximum output state for any iteration. This, however, does not yet address the case where the loop is not entered at all. Therefore, we need to take the least-upper bound of the start state and the result of the analysis of the worst-case iteration.
To overestimate time-dependent energy usage, we must revert component timestamps to the time of entering the loop. So, if a component is switched to a greater state at some point in the loop, the analysis assumes it has been in this state since entering the loop. Note that the least-upper bound of energy-aware component states does exactly this: maximise the non-energy-aware component state and minimise the timestamp. Hence, if we take $\Gamma_{base} = \mathbf{lub}(\Gamma_{in}, \Gamma_{out})$ we find both the maximum output states and the minimum timestamps.
Now, we can add the consumption of the loop itself. We perform the following calculation for each component: $C_i^{\Gamma_{base}} :: \mathfrak{e} = C_i^{\Gamma_{in}} :: \mathfrak{e} + (C_i^{\Gamma_{out}} :: \mathfrak{e} - C_i^{\Gamma_{in}} :: \mathfrak{e}) \cdot ib$. We do something similar for the time consumption: $\mathfrak{t}_{ret} = \mathfrak{t} + ((\mathfrak{t}' - \mathfrak{t}) \cdot ib)$.

5. Processing time-dependent energy function **process-td**. When analysing an iteration of a loop, we must take care not to include any energy consumption outside of the iteration. This would lead to a large overestimation, since it would be multiplied by the (possibly overestimated) number of iterations. Therefore, before analysing the body, we add the time-dependent energy consumption to the energy accumulator for each component and set all timestamps to the current time. Otherwise, the time-dependent consumption before entering the loop would also be included in the analysis of the iteration. We introduce the function **process-td**$(\Gamma, \mathfrak{t})$, which adds $td(C_i, \mathfrak{t})$ to $C_i :: \mathfrak{e}$ and sets $C_i :: \tau$ to $\mathfrak{t}$, for each component C_i in Γ.

Applying the rules gives an overestimation of the sum of the incidental energy consumption and the time-dependent energy consumption. However, the time-dependent energy consumption is only added to the accumulator at changes of component states. So, as for the energy-aware semantics, the time the components are in their current state should still be accounted for by calculating $\mathfrak{e}_{system}(\Gamma_{end}, \mathfrak{t}_{end})$.

3.4 Building Hardware Models

The various elements of a component model that need to be defined by the builder of such a model are listed in Table 1.

	Function	**Default values**
$C_i :: s$	Component state. Elements are integers.	Empty by default. Elements are initialized by 0 by default.
$C_i :: \mathfrak{E}_f$	For every component function f. Incidental energy usage for a call to f.	$C_i :: \mathfrak{E}_f = 0$
$C_i :: \mathfrak{T}_f$	For every component function f. Time consumption for a call to f.	$C_i :: \mathfrak{T}_f = 0$
$C_i :: \delta_f(s)$	For every component function f. Component state update function for a call to f.	$C_i :: \delta_f(s) = s$
$C_i :: \phi(s)$	Computes the (constant) power draw while in state s.	$C_i :: \phi(s) = 0$

Table 1. The elements of a component model C_i that need to be defined by the builder of such a model.

3.5 Motivation

In this section we provide further motivation for several important decisions in the design of the modeling and logic. First, we explain why power draw should be constant for each component state, by first showing why it should not increase, then why it should not decrease. Second, we explain why, in the (aWhile) rule, we cannot simply analyse one iteration, but need to find the fixpoint first.

Constant power draw We have assumed that while in a certain state, a component has a constant power draw. Alternatively, we have considered that switching to a certain state could signify the start-point of a certain power-draw function. Such a function could express a power draw that, from the moment of changing to a certain state, increases or decreases over time. For instance, one could imagine that turning a component on will use a lot of power at first, then decrease and converge to a constant, lower power usage[4].

Consider the following program:

if e **then**
  ⟨ change to state s_2 ⟩
else
  ⟨ do nothing ⟩
end if

Suppose we have a component C that has just two states $s_2 > s_1$. At the beginning of the conditional the component C is in state s_1. In the then-branch a function of C is executed that raises its state to s_2, while in the else-branch the state is not changed. In the analysis, we use production rule (aIf) and take the state that represents the least upper bound of both branches. Hence for the analysis, the worst case is when we are switching the state from s_1 to s_2.

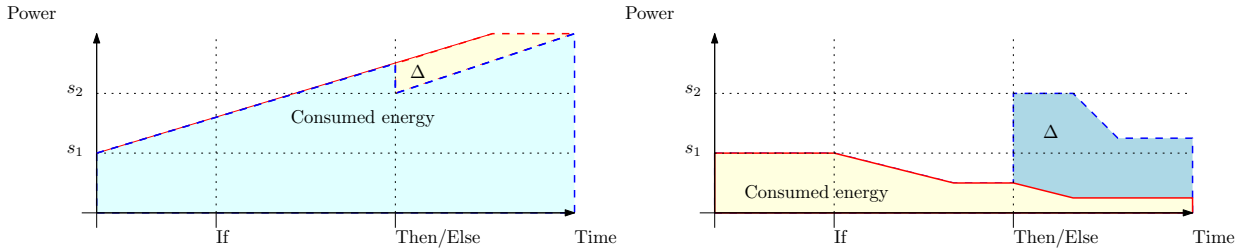

Fig. 4. Increasing and decreasing power consumption in a conditional.

Fig. 4 shows the energy usage for the example. On the left, power draw is increasing. On the right, power draw is decreasing. In the case where, after changing to a certain state, power draw increases over time (left), we see that after some time, the power consumed in state s_1 exceeds the power consumed initially after changing to state s_2. This means that it is not possible to overestimate power draw by taking the least upper bound of component states. This is why we only allow non-increasing power draw. In the case where power draw decreases after switching to state s_2 (right), taking the least upper bound still gives a correct (but possibly large) overestimation.

Now consider the following program:

while e **do**
  ⟨ change to state s_3 ⟩
  ⟨ change to state s_2 ⟩
  ⟨ change to state s_3 ⟩
end while

Suppose we have a component C with three states $s_3 > s_2 > s_1$. Before the loop, the component C is in state s_1. Inside the loop, the program switches first to s_3, then to s_2, then to s_3 again. In the analysis, we use production rule (aWhile) which calculates the maximum state, uses this to calculate the maximum energy consumption within one cycle, then overestimates the total energy consumption by multiplying the maximum consumption within a cycle by the (overestimated) number of cycles (i.e. the ranking function) and assuming that the component was in the maximal state from the time at which the loop was entered. This last part is where decreasing power draw poses a problem.

Fig. 5 shows the energy usage for the example. On the left, power draw is increasing. On the right, power draw is decreasing. We see that in the case where power draw increases over time (left), changing to the

[4] Notice that such a situation may be modeled in the current modeling by coupling a high incidental energy-usage with a low constant power draw

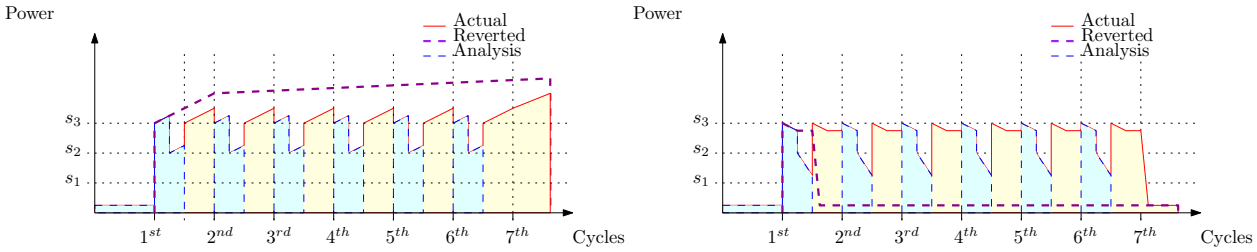

Fig. 5. Increasing and decreasing power consumption in a while.

maximum state after any cycle and reverting timestamps to their values before entering the loop correctly overestimates energy consumption. On the right we see a power draw function that decreases quickly, then converges to some low constant value. In this case, taking the maximum state and reverting timestamps leads to an underestimation. In other words, power draw should be non-decreasing.

Since we showed earlier that power draw should be non-increasing, we can conclude: component models should have a constant power draw for each component state.

Why we need the fixpoint function for analysing while loops The production rule ($aWhile$) has to deal with the problem that every loop cycle could consume a different amount of energy. Even if the power consumption is constant for each state and even if we separate the incidental energy consumption from the time-dependent energy consumption, there is still another problem to deal with.

Consider the following program:

while e **do**
  ⟨ raise the state ⟩
  ⟨ raise the state ⟩
  ⟨ change to state 2 ⟩
end while

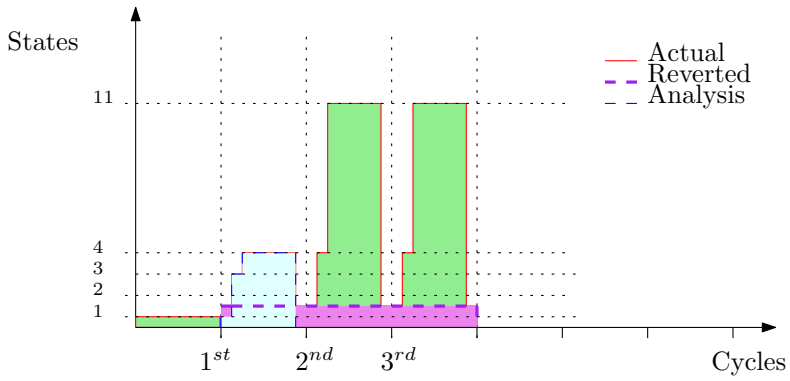

Fig. 6. Possible execution of the example program.

In Fig. 6 we show a possible execution this program, wtih three loop cycles. Component C is in state 1 when the loop is entered. Then, in the first iteration, the state is raised to 3 and then 4. Finally, the state is lowered to 2. In the second iteration, the state is raised to 4, then 11, which consumes a large amount of energy. Finally, the state is again lowered to 2. Since we enter the loop with the same state now as the previous time, the results are the same.

We see that the energy-usage of a loop cycle depends on the component states. We must therefore apply the fixpoint function and use the state which represents the least-upper bound of all possible states upon entry of a cycle to find the maximum consumption within one cycle.

4 Example: Wireless Sensor Node

As an example for our analysis, we have modeled a wireless sensor node with a temperature sensor and a radio, then calculated energy usage of two variants of a program that takes a series of measurements and sends them over the radio.

4.1 Modeling

The wireless sensor node consists of the component for implicit consumption C_{imp}, a temperature sensor C_s and the radio C_r.

Implicit resource consumption The component C_{imp} is part of every system model. The model used in this example is given in Table 2. A stateless C_{imp} is assumed that does not consume energy based on time. As there is no state, the model consists of a set of constants.

Constants	$C_{imp} :: \mathfrak{T}_e = 10,\ C_{imp} :: \mathfrak{T}_a = 5,\ C_{imp} :: \mathfrak{T}_w = 25,\ C_{imp} :: \mathfrak{T}_{ite} = 25,$ $C_{imp} :: \mathfrak{E}_e = 10,\ C_{imp} :: \mathfrak{E}_a = 5,\ C_{imp} :: \mathfrak{E}_w = 25,\ C_{imp} :: \mathfrak{E}_{ite} = 25$

Table 2. The C_{imp} component model.

Sensor Component model C_s represents the sensor. It is given in Table 3. The sensor itself implements a single function: m, which takes a measurement. Therefore, there is also a single time-usage constant and a single energy-usage constant. In this analysis we are not interested in the return value, so we define it as always 0. The sensor cannot be turned off. It has no state. Therefore, it has a constant power-draw, which is set to 3 in the model.

Constants	$C_s :: \mathfrak{T}_m = 10,\ C_s :: \mathfrak{E}_m = 40$
Return value	$C_s :: rv_m(\_) = 0$
Power draw	$C_s :: \phi(\_) = 3$

Table 3. The C_s component model.

Radio Component model C_r represents the radio. It is shown in Table 4. The radio implements four functions: *on*, *off*, *queue* and *send*. The *queue* function queues a measurement for transmission, *send* transmits all the measurements in the queue. It is possible to queue measurements when the radio is turned off. The state of C_r consists of a single variable $C_r :: s.on$ (which is one/true if the radio is turned on and zero/false when it is off). There are constants in the component model representing the costs of turning the radio on or off, putting a measurement in the queue and for sending all the measurements in the queue.

Notice that since in our modeling, incidental energy consumption may not depend on the component state, sending the queue takes a constant amount of energy. Of course, in reality the energy cost of sending all the measurements in the queue *does* depend on the length of the queue. This is modeled by amortising the extra costs for sending a queue with one extra element to the incidental energy costs of queueing a measurement. In this analysis we are not interested in the return values of the component functions, so we define them as always 0.

State	$C_r :: s = \{on : [0, 1]\}$
Constants	$C_r :: \mathfrak{T}_{on} = 400,\ C_r :: \mathfrak{T}_{off} = 200,\ C_r :: \mathfrak{T}_{queue} = 30,\ C_r :: \mathfrak{T}_{send} = 100,$ $C_r :: \mathfrak{E}_{on} = 400,\ C_r :: \mathfrak{E}_{off} = 200,\ C_r :: \mathfrak{E}_{queue} = 30,\ C_r :: \mathfrak{E}_{send} = 100$
Update functions	$C_r :: \delta_{on}(s) = s[on \leftarrow 1], C_r :: \delta_{off}(s) = s[on \leftarrow 0]$
Return values	$C_r :: rv_{on}(\_) = 0, C_r :: rv_{off}(\_) = 0,$ $C_r :: rv_{queue}(\_) = 0, C_r :: rv_{send}(\_) = 0$
Power draw	$C_r :: \phi(s) = 2 + 200 \cdot s.on$

Table 4. The C_r component model.

4.2 Analysis

We will compare two implementations of an algorithm that takes n measurements with the sensor and sends the results over the radio. The AlwaysOn algorithm sends the results of all measurements out individually. The Buffering algorithm collects the results of 10 measurements and sends them in a single message. The latter is an example of *duty-cycling*, which is a well-known method for energy conservation in wireless sensor networks [25]. Our analysis indeed gives the expected result: the Buffering is more energy-efficient.

As the symbolic state environment ρ is not used in the example (it is used in function calls only), we have omitted it for readability. Also, we use an additional rule for component function calls:

$$\frac{\Gamma_1 = \Gamma[C_i :: \mathfrak{e} \mathrel{+}= C_i :: \mathfrak{E}_f] \quad C_i :: s \equiv C_i :: \delta_f(C_i :: s)}{\{\Gamma; \mathfrak{t}\} C_i :: f(args) \{\Gamma_1; \mathfrak{t} + C_i :: \mathfrak{T}_f\}} (\text{aCallCmpFinc})$$

If a component function does not alter the state, time-dependent energy consumption does not have to be processed. As we use a symbolic start state, this simplifies presentation of the example a lot.

Example AlwaysOn:

$C_r :: on();$
while$_N$ $N > 0$ **do**
  $M = C_s :: m();$
  $C_r :: queue(M);$
  $C_r :: send();$
  $N = N - 1$
end while;
$C_r :: off();$

Example Buffering:

while$_{\lceil N/10 \rceil}$ $N > 0$ **do**
  $I = 10;$
  while$_{10}$ $I > 0$ **do**
    $M = C_s :: m();$
    $C_r :: queue(M);$
    $I = I - 1;$
    $N = N - 1$
  end while;
  $C_r :: on();$
  $C_r :: send();$
  $C_r :: off()$
end while

Analysis of the AlwaysOn Algorithm The evaluation of the states in the application of the logic to the AlwaysOn algorithm is shown in Table 5. We start with a state Γ_s in which every variable is a symbol. We first apply the C rule on the concatenation. Then, for the left branch we can apply the component function application rule (aCallCmpF) for $C_r :: on()$. This sets the state of the radio to "on", updates its timestamp and adds the time-dependent energy consumption. Also, the incidental energy and time usage are added.

We apply the (aConcat) rule again, then continue with the analysis of the while loop. First, we must apply the **process-td** function, which results in Γ_2. It sets the timestamps to the current time and adds time-dependent consumption for the radio ($400 \cdot (2 + 200 \cdot 1) = 80800$) and the sensor ($(\mathfrak{t}_0 + 400 - \tau_{\mathbf{0}}^{\mathbf{s}}) \cdot 3$). We then apply the **wci** function. The loop body does not change the component states. Therefore, the worst-case iteration function reduces to the identity function here: it returns Γ_2. We now analyse the guard. The variable (aVar) and constant (aConst) rules do not affect energy or time consumption. The rule for evaluating an expression (aBinOp) adds the constant time and energy costs for the implicit component. Then, the constant incidental resource consumption is added for the while loop.

We then analyse the loop body, which starts with a sensor measurement. As this component function does not change the state, we can apply the (aCallCmpFinc) rule. Only incidental energy (40) and time (10) consumption are added. After the method call, the return value is assigned, so the corresponding constants are added to the global time and the energy consumption of the implicit component.

After that, the measurement is queued. This does not change the state of the radio. So, again, we can apply the (aCallCmpFinc) rule, which only adds the incidental consumption. After sending the queue, N is decreased by one. In the analysis we subsequently add the constant costs for evaluation of an expression and assignment to the time and the energy consumption.

$$\mathbf{P_{guard}}: \quad \frac{\overline{\{\Gamma_2;\mathfrak{t}_1\}N\{\Gamma_2;\mathfrak{t}_1\}}(\text{aVar}) \quad \overline{\{\Gamma_2;\mathfrak{t}_1\}0\{\Gamma_2;\mathfrak{t}_1\}}(\text{aConst}) \quad \Gamma_3 = \Gamma_2[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_e]}{\{\Gamma_2;\mathfrak{t}_1\}N > 0\{\Gamma_3;\mathfrak{t}_2 = \mathfrak{t}_1 + C_{imp}::\mathfrak{T}_e\}}(\text{aBinOp})$$

$$\mathbf{P_m}: \quad \frac{\frac{\Gamma_5{=}\Gamma_4[C_s::\mathfrak{e} \mathrel{+}= C_s::\mathfrak{E}_m] \quad C_s::s \equiv C_s::\delta_m(C_s::s)}{\{\Gamma_4;\mathfrak{t}_3\}C_s::m()\{\Gamma_5;\mathfrak{t}_4 = \mathfrak{t}_3 + C_s::\mathfrak{T}_m\}}(\text{aCallCmpFinc}) \quad \Gamma_6{=}\Gamma_5[C_{imp}::\mathfrak{e}{+}{=}C_{imp}::\mathfrak{E}_a]}{\{\Gamma_4;\mathfrak{t}_3\}M = C_s::m()\{\Gamma_6;\mathfrak{t}_5 = \mathfrak{t}_4 + C_{imp}::\mathfrak{T}_a\}}(\text{aAssign})$$

$$\mathbf{P_{queue}}: \quad \frac{\Gamma_7 = \Gamma_6[C_r::\mathfrak{e} \mathrel{+}= C_r::\mathfrak{E}_{queue}] \quad C_r::s \equiv C_r::\delta_{queue}(C_r::s)}{\{\Gamma_6;\mathfrak{t}_5\}C_r::queue(M)\{\Gamma_7;\mathfrak{t}_6 = \mathfrak{t}_5 + C_r::\mathfrak{T}_{queue}\}}(\text{aCallCmpFinc})$$

$$\mathbf{P_{send}}: \quad \frac{\Gamma_8 = \Gamma_7[C_r::\mathfrak{e} \mathrel{+}= C_r::\mathfrak{E}_{send}] \quad C_r::s \equiv C_r::\delta_{send}(C_r::s)}{\{\Gamma_7;\mathfrak{t}_6\}C_r::send()\{\Gamma_8;\mathfrak{t}_7 = \mathfrak{t}_6 + C_r::\mathfrak{T}_{send}\}}(\text{aCallCmpFinc})$$

$$\mathbf{P_{n-1}}: \quad \frac{\overline{\{\Gamma_8;\mathfrak{t}_7\}N\{\Gamma_8;\mathfrak{t}_7\}}(\text{aVar}) \quad \overline{\{\Gamma_8;\mathfrak{t}_7\}1\{\Gamma_8;\mathfrak{t}_7\}}(\text{aConst}) \quad \Gamma_9 = \Gamma_8[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_e]}{\{\Gamma_8;\mathfrak{t}_7\}N - 1\{\Gamma_9;\mathfrak{t}_8 = \mathfrak{t}_7 + C_{imp}::\mathfrak{T}_e\}}(\text{aBinOp})$$

$$\mathbf{P_{n--}}: \quad \frac{\mathbf{P_{n-1}} \quad \Gamma_{10} = \Gamma_9[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_a]}{\{\Gamma_8;\mathfrak{t}_7\}N = N - 1\{\Gamma_{10};\mathfrak{t}_9 = \mathfrak{t}_8 + C_{imp}::\mathfrak{T}_a\}}(\text{aAssign})$$

$$\mathbf{P_{body}}: \quad \frac{\mathbf{P_m} \quad \frac{\mathbf{P_{queue}} \quad \frac{\mathbf{P_{send}} \quad \mathbf{P_{n--}}}{\{\Gamma_7;\mathfrak{t}_6\}C_r::send(); N = N - 1\{\Gamma_{10};\mathfrak{t}_9\}}(\text{aConcat})}{\{\Gamma_6;\mathfrak{t}_5\}C_r::queue(M); C_r::send(); N = N - 1\{\Gamma_{10};\mathfrak{t}_9\}}(\text{aConcat})}{\{\Gamma_4;\mathfrak{t}_3 = \mathfrak{t}_2 + C_{imp}::\mathfrak{T}_w\}M = C_s::m(); C_r::queue(M); C_r::send(); N = N - 1\{\Gamma_{10};\mathfrak{t}_9\}}(\text{aConcat})$$

$$\mathbf{P_{while}}: \quad \frac{\Gamma_2 = \textbf{process-td}(\Gamma_1,\mathfrak{t}_1) \quad \mathbf{P_{guard}} \quad \Gamma_4 = \Gamma_3[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_w] \quad \mathbf{P_{body}}}{\{\Gamma_1;\mathfrak{t}_1\}\textbf{while}_N\ N > 0\ \textbf{do}\ \ldots\ \textbf{end while}\{\Gamma_{11};\mathfrak{t}_{10}\}}(\text{aWhile})$$

$$\mathbf{P_{while-off}}: \quad \frac{\mathbf{P_{while}} \quad \frac{\Gamma_{12} = \Gamma_{11}[C_r::s \leftarrow C_r::\delta_{off}(C_r::s), C_r::\tau \leftarrow \mathfrak{t}_{10}, C_r::\mathfrak{e} \mathrel{+}= C_r::\mathfrak{E}_{off} + td(C_r,\mathfrak{t}_{10})]}{\{\Gamma_{11};\mathfrak{t}_{10}\}C_r::off()\{\Gamma_{12};\mathfrak{t}_{11} = \mathfrak{t}_{10} + C_r::\mathfrak{T}_{off}\}}(\text{aCallCmpF})}{\{\Gamma_1;\mathfrak{t}_1\}\ldots;C_r::off();\{\Gamma_{12};\mathfrak{t}_{11}\}}(\text{aConcat})$$

$$\textbf{START}: \quad \frac{\Gamma_1 = \Gamma_0[C_r::s \leftarrow C_r::\delta_{on}(C_r::s), C_r::\tau \leftarrow \mathfrak{t}_0, C_r::\mathfrak{e} \mathrel{+}= C_r::\mathfrak{E}_{on} + td(C_r,\mathfrak{t}_0)]}{\{\Gamma_s;\mathfrak{t}_0\}C_r::on()\{\Gamma_1;\mathfrak{t}_1 = \mathfrak{t}_0 + C_r::\mathfrak{T}_{on}\}}(\text{aCallCmpF}) \quad \mathbf{P_{while-off}}}{\{\Gamma_s;\mathfrak{t}_0\}C_r.on();\ldots;C_r::off();\{\Gamma_{12};\mathfrak{t}_{11}\}}(\text{aConcat})$$

Fig. 7. Analysis of the AlwaysOn algorithm, $\mathbf{P}_*$ represent partial proofs, i.e. they refer to other parts of the figure.

We now have to overestimate, from the evaluation of the loop body, the state after the whole loop. This is done using the $\mathbf{oe}(\Gamma_2,\mathfrak{t}_1,\Gamma_{10},\mathfrak{t}_9,N)$ function, which results in Γ_{11} and $\mathfrak{t}_{10}$. This looks at the consumption of the body, by substracting the values in Γ_2 from those in Γ_{10} and adding the product of the difference with the loop bound N. The same is done for time: the difference between $\mathfrak{t}_{10}$ and $\mathfrak{t}_1$, 195, multiplied by N is added to $\mathfrak{t}_1$. The **oe** function also over-estimates the output state of the loop. Since here the loop body does not alter the state, this is the same as the input state (the radio is on).

Finally, we turn the radio off. This means that the time-dependent energy for the radio is added and the costs of $C_r::off()$ must be added. The time-dependent energy consumption is $195 \cdot N \cdot 202 = 39390 \cdot N$, the time costs are 100 and the incidental energy cost is 100.

After applying the Hoare logic, we still need to add the time-dependent energy-consumption for each component. The implicit component does not consume energy based on time (although it would be simple to add this if needed). Its energy consumption at Γ_{12} is thus final. The result for the sensor is $\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1200 + 40 \cdot N + (200 + 195 \cdot N) \cdot 3 = \mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1800 + 625 \cdot N$. The result for the radio is $\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81400 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 39520 \cdot N + 200 \cdot 2 = \mathfrak{e}_\mathbf{0}^\mathbf{r} + 81800 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 39520 \cdot N$.

Finally, we can calculate the sum of the energy-usage of all components:

$$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 55 \cdot N + \mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1800 + 625 \cdot N + \mathfrak{e}_\mathbf{0}^\mathbf{r} + 81800 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 39520 \cdot N =$$

$$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + \mathfrak{e}_\mathbf{0}^\mathbf{s} + \mathfrak{e}_\mathbf{0}^\mathbf{r} + 83600 + 40200 \cdot N + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$$

State	$\mathfrak{t}$	$C_{imp} :: \mathfrak{e}$	$C_s :: \tau$	$C_s :: \mathfrak{e}$	$C_r :: s.on$	$C_r :: \tau$	$C_r :: \mathfrak{e}$
$\Gamma_s, \mathfrak{t}_0$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\tau_\mathbf{0}^\mathbf{s}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_1, \mathfrak{t}_1$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\tau_\mathbf{0}^\mathbf{s}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 400 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_2, \mathfrak{t}_1$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1200$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81200 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_3, \mathfrak{t}_2$	$\mathfrak{t}_\mathbf{0} + 410$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 10$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1200$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81200 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_4, \mathfrak{t}_3$	$\mathfrak{t}_\mathbf{0} + 435$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 35$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1200$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81200 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_5, \mathfrak{t}_4$	$\mathfrak{t}_\mathbf{0} + 445$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 35$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1240$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81200 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_6, \mathfrak{t}_5$	$\mathfrak{t}_\mathbf{0} + 450$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1240$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81200 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_7, \mathfrak{t}_6$	$\mathfrak{t}_\mathbf{0} + 480$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1240$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81230 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_8, \mathfrak{t}_7$	$\mathfrak{t}_\mathbf{0} + 580$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1240$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81330 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_9, \mathfrak{t}_8$	$\mathfrak{t}_\mathbf{0} + 590$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 50$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1240$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81330 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_{10}, \mathfrak{t}_9$	$\mathfrak{t}_\mathbf{0} + 595$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 55$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1240$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81330 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_{11}, \mathfrak{t}_{10}$	$\mathfrak{t}_\mathbf{0} + 400 + 195 \cdot N$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 55 \cdot N$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1200 + 40 \cdot N$	1	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81200 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 130 \cdot N$
$\Gamma_{12}, \mathfrak{t}_{11}$	$\mathfrak{t}_\mathbf{0} + 600 + 195 \cdot N$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 55 \cdot N$	$\mathfrak{t}_\mathbf{0} + 400$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_0 - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 1200 + 40 \cdot N$	0	$\mathfrak{t}_\mathbf{0} + 400 + 195 \cdot N$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 81400 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 39520 \cdot N$

Table 5. Global states for the AlwaysOn algorithm.

This constitutes an energy usage that is symbolic over both the input variables of the program and starting state of the components. If we assume an "empty" starting state, in which the radio has just been turned off and no energy has been consumed yet, then this results in:

$$83600 + 40200 \cdot N$$

Analysis of the Buffering Algorithm The evaluation of the states in the application of the logic to the Buffering algorithm is shown in Table 6. We start with a state Γ_s again, in which every variable is a symbol.

The first rule to apply is the (aWhile) rule, for which we start with **process-td**. Analysis of the loop guard $N > 0$ adds the constant costs for evaluating an expression and also the incidental costs of loop itself are added.

We can now evaluate the body of the outer loop. We start by adding the constant costs for the assignment. Then we move on to the inner loop. This again means processing time-dependent energy consumption first. Then, we add the constant costs for evaluating an expression and for the while loop, respectively. Moving on to the inner body, we start by analysing the sensor measurement. Again, since the sensor has no state, we do not have to update a timestamp or add time-dependent energy consumption. At the end of the loop body, both the counters are decreased. This adds the constant costs for expression evaluation and assignment twice.

Now, we need to overestimate the state after the inner loop. We take state 5 and add the difference with state 14, multiplied by the bound 10. For instance, for global time, this adds $110 \cdot 10 = 1100$ to $\mathfrak{t}_3$.

After the inner loop, the series of measurements is sent over the radio. First, the radio is turned on. This changes the state of the radio, so we also need to update its timestamp and calculate time-dependent energy onsumption. Then, the constant costs for sending the queue are added. When turning the radio off, again, the state is changed, so the timestamp for the radio component is also updated and time-dependent energy

$\mathbf{P_{outerguard}}$:

$$\frac{\overline{\{\Gamma_1;\mathfrak{t}_0\}N\{\Gamma_1;\mathfrak{t}_0\}}^{(\text{aVar})} \quad \overline{\{\Gamma_1;\mathfrak{t}_0\}0\{\Gamma_1;\mathfrak{t}_0\}}^{(\text{aConst})} \quad \Gamma_1 = \Gamma_0[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma_1;\mathfrak{t}_0\}N > 0\{\Gamma_2;\mathfrak{t}_1 = \mathfrak{t}_0 + C_{imp} :: \mathfrak{T}_e\}}^{(\text{aBinOp})}$$

$\mathbf{P_i}$:

$$\frac{\overline{\{\Gamma_3;\mathfrak{t}_2\}10\{\Gamma_3;\mathfrak{t}_2\}}^{(\text{aConst})} \quad \Gamma_4 = \Gamma_3[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\{\Gamma_3;\mathfrak{t}_2\}I = 10\{\Gamma_4;\mathfrak{t}_3 = \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_a\}}^{(\text{aAssign})}$$

$\mathbf{P_{innerguard}}$:

$$\frac{\overline{\{\Gamma_5;\mathfrak{t}_3\}I\{\Gamma_5;\mathfrak{t}_3\}}^{(\text{aVar})} \quad \overline{\{\Gamma_5;\mathfrak{t}_3\}0\{\Gamma_5;\mathfrak{t}_3\}}^{(\text{aConst})} \quad \Gamma_6 = \Gamma_5[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma_5;\mathfrak{t}_3\}I > 0\{\Gamma_6;\mathfrak{t}_4 = \mathfrak{t}_3 + C_{imp} :: \mathfrak{T}_e\}}^{(\text{aBinOp})}$$

$\mathbf{P_m}$:

$$\frac{\Gamma_8{=}\Gamma_7[C_s :: \mathfrak{e} \mathrel{+}= C_s :: \mathfrak{E}_m] \quad C_s :: s \equiv C_s :: \delta_m(C_s :: s)}{\{\Gamma_7;\mathfrak{t}_5\}C_s :: m()\{\Gamma_8;\mathfrak{t}_6 = \mathfrak{t}_5 + C_s :: \mathfrak{T}_m\}}^{(\text{aCallCmpFinc})} \quad \frac{}{\{\Gamma_7;\mathfrak{t}_5\}M = C_s :: m()\{\Gamma_9;\mathfrak{t}_7 = \mathfrak{t}_6 + C_{imp} :: \mathfrak{T}_a\}}^{(\text{aAssign})} \quad \Gamma_9{=}\Gamma_8[C_{imp} :: \mathfrak{e}{+}{=}C_{imp} :: \mathfrak{E}_a]$$

$\mathbf{P_{queue}}$:

$$\frac{\Gamma_{10} = \Gamma_9[C_r :: \mathfrak{e} \mathrel{+}= C_r :: \mathfrak{E}_{queue}] \quad C_r :: s \equiv C_r :: \delta_{queue}(C_r :: s)}{\{\Gamma_9;\mathfrak{t}_7\}C_r :: queue(M)\{\Gamma_{10};\mathfrak{t}_8 = \mathfrak{t}_7 + C_r :: \mathfrak{T}_{queue}\}}^{(\text{aCallCmpFinc})}$$

$\mathbf{P_{i-1}}$:

$$\frac{\overline{\{\Gamma_{10};\mathfrak{t}_8\}I\{\Gamma_{10};\mathfrak{t}_8\}}^{(\text{aVar})} \quad \overline{\{\Gamma_{10};\mathfrak{t}_8\}1\{\Gamma_{10};\mathfrak{t}_8\}}^{(\text{aConst})} \quad \Gamma_{11} = \Gamma_{10}[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma_{10};\mathfrak{t}_8\}I - 1\{\Gamma_{11};\mathfrak{t}_9 = \mathfrak{t}_8 + C_{imp} :: \mathfrak{T}_e\}}^{(\text{aBinOp})}$$

$\mathbf{P_{i--}}$:

$$\frac{\mathbf{P_{i-1}} \quad \Gamma_{12} = \Gamma_{11}[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\{\Gamma_{10};\mathfrak{t}_8\}I = I - 1\{\Gamma_{12};\mathfrak{t}_{10} = \mathfrak{t}_9 + C_{imp} :: \mathfrak{T}_a\}}^{(\text{aAssign})}$$

$\mathbf{P_{n-1}}$:

$$\frac{\overline{\{\Gamma_{12};\mathfrak{t}_{10}\}N\{\Gamma_{12};\mathfrak{t}_{10}\}}^{(\text{aVar})} \quad \overline{\{\Gamma_{12};\mathfrak{t}_{10}\}1\{\Gamma_{12};\mathfrak{t}_{10}\}}^{(\text{aConst})} \quad \Gamma_{13} = \Gamma_{12}[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma_{12};\mathfrak{t}_{10}\}N - 1\{\Gamma_{13};\mathfrak{t}_{11} = \mathfrak{t}_{10} + C_{imp} :: \mathfrak{T}_e\}}^{(\text{aBinOp})}$$

$\mathbf{P_{n--}}$:

$$\frac{\mathbf{P_{n-1}} \quad \Gamma_{14} = \Gamma_{13}[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\{\Gamma_{12};\mathfrak{t}_{10}\}N = N - 1\{\Gamma_{14};\mathfrak{t}_{12} = \mathfrak{t}_{11} + C_{imp} :: \mathfrak{T}_a\}}^{(\text{aAssign})}$$

$\mathbf{P_{innerbody}}$:

$$\frac{\mathbf{P_m} \quad \dfrac{\mathbf{P_{queue}} \quad \dfrac{\mathbf{P_{i--}} \quad \mathbf{P_{n--}}}{\{\Gamma_{10};\mathfrak{t}_8\}I = I - 1; N = N - 1\{\Gamma_{14};\mathfrak{t}_{12}\}}^{(\text{aConcat})}}{\{\Gamma_9;\mathfrak{t}_7\}C_r :: queue(M); I = I - 1; N = N - 1\{\Gamma_{14};\mathfrak{t}_{12}\}}^{(\text{aConcat})}}{\{\Gamma_7;\mathfrak{t}_5 = \mathfrak{t}_4 + C_{imp} :: \mathfrak{T}_w\}M = C_s :: m(); C_r :: queue(M); I = I - 1; N = N - 1\{\Gamma_{14};\mathfrak{t}_{12}\}}^{(\text{aConcat})}$$

$\mathbf{P_{inner}}$:

$$\frac{\Gamma_5 = \textbf{process-td}(\Gamma_4, \mathfrak{t}_3) \quad \mathbf{P_{innerguard}} \quad \Gamma_7 = \Gamma_6[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \mathbf{P_{innerbody}}}{\{\Gamma_4;\mathfrak{t}_3\}\textbf{while}_{10}\ I > 0\ \textbf{do}\ \ldots\ \textbf{end while}\{\Gamma_{15};\mathfrak{t}_{13}\}}^{(\text{aWhile})}$$

$\mathbf{P_{send}}$:

$$\frac{\Gamma_{17} = \Gamma_{16}[C_r :: \mathfrak{e} \mathrel{+}= C_r :: \mathfrak{E}_{send}] \quad C_r :: s \equiv C_r :: \delta_{send}(C_r :: s)}{\{\Gamma_{16};\mathfrak{t}_{14}\}C_r :: send()\{\Gamma_{17};\mathfrak{t}_{15} = \mathfrak{t}_{14} + C_r :: \mathfrak{T}_{send}\}}^{(\text{aCallCmpFinc})}$$

$\mathbf{P_{radio2}}$:

$$\frac{\mathbf{P_{send}} \quad \dfrac{\Gamma_{18} = \Gamma_{17}[C_r :: s \leftarrow C_r :: \delta_{off}(C_r :: s), C_r :: \tau \leftarrow \mathfrak{t}_{15}, C_r :: \mathfrak{e} \mathrel{+}= C_r :: \mathfrak{E}_{off} + td(C_r, \mathfrak{t}_{15})]}{\{\Gamma_{17};\mathfrak{t}_{15}\}C_r :: off()\{\Gamma_{18};\mathfrak{t}_{16} = \mathfrak{t}_{15} + C_r :: \mathfrak{T}_{off}\}}^{(\text{aCallCmpF})}}{\{\Gamma_{16};\mathfrak{t}_{14}\}C_r :: send(); C_r :: off()\{\Gamma_{18};\mathfrak{t}_{16}\}}^{(\text{aConcat})}$$

$\mathbf{P_{radio}}$:

$$\frac{\dfrac{\Gamma_{16} = \Gamma_{15}[C_r :: s \leftarrow C_r :: \delta_{on}(C_r :: s), C_r :: \tau \leftarrow \mathfrak{t}_{13}, C_r :: \mathfrak{e} \mathrel{+}= C_r :: \mathfrak{E}_{on} + td(C_r, \mathfrak{t}_{13})]}{\{\Gamma_{15};\mathfrak{t}_{13}\}C_r :: on()\{\Gamma_{16};\mathfrak{t}_{14} = \mathfrak{t}_{13} + C_r :: \mathfrak{T}_{on}\}}^{(\text{aCallCmpF})} \quad \mathbf{P_{radio2}}}{\{\Gamma_{15};\mathfrak{t}_{13}\}C_r :: on(); C_r :: send(); C_r :: off()\{\Gamma_{18};\mathfrak{t}_{16}\}}^{(\text{aConcat})}$$

$\mathbf{P_{outerbody}}$:

$$\frac{\mathbf{P_i} \quad \dfrac{\mathbf{P_{inner}} \quad \mathbf{P_{radio}}}{\{\Gamma_4;\mathfrak{t}_3\}\textbf{while}_{10}\ I > 0\ \textbf{do}\ \ldots\ \textbf{end while}; \ldots \{\Gamma_{18};\mathfrak{t}_{16}\}}^{(\text{aConcat})}}{\{\Gamma_3;\mathfrak{t}_2 = \mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_w\}I = 10; \textbf{while}_{10}\ I > 0\ \textbf{do}\ \ldots\ \textbf{end while}; \ldots \{\Gamma_{18};\mathfrak{t}_{16}\}}^{(\text{aConcat})}$$

START :

$$\frac{\Gamma_1 = \textbf{process-td}(\Gamma_s, \mathfrak{t}_0) \quad \mathbf{P_{outerguard}} \quad \Gamma_3 = \Gamma_2[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \mathbf{P_{outerbody}}}{\{\Gamma_s;\mathfrak{t}_0\}\textbf{while}_N\ N > 0\ \textbf{do}\ \ldots\ \textbf{end while}\{\Gamma_{19};\mathfrak{t}_{17}\}}^{(\text{aWhile})}$$

Fig. 8. Analysis of the Buffering algorithm, $\mathbf{P}_*$ represent partial proofs, i.e. they refer to other parts of the figure.

State	$\mathfrak{t}$	$C_{imp} :: \mathfrak{e}$	$C_s :: \tau$	$C_s :: \mathfrak{e}$	$C_r :: s.on$	$C_r :: \tau$	$C_r :: \mathfrak{e}$
$\Gamma_s, \mathfrak{t}_0$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\tau_\mathbf{0}^\mathbf{s}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_1, \mathfrak{t}_0$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s}) \cdot 3$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_2, \mathfrak{t}_1$	$\mathfrak{t}_\mathbf{0} + 10$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 10$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s}) \cdot 3$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_3, \mathfrak{t}_2$	$\mathfrak{t}_\mathbf{0} + 35$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 35$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s}) \cdot 3$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_4, \mathfrak{t}_3$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s}) \cdot 3$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_5, \mathfrak{t}_3$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_6, \mathfrak{t}_4$	$\mathfrak{t}_\mathbf{0} + 50$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 50$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_7, \mathfrak{t}_5$	$\mathfrak{t}_\mathbf{0} + 75$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_8, \mathfrak{t}_6$	$\mathfrak{t}_\mathbf{0} + 85$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_9, \mathfrak{t}_7$	$\mathfrak{t}_\mathbf{0} + 90$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 80$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_{10}, \mathfrak{t}_8$	$\mathfrak{t}_\mathbf{0} + 120$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 80$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 30$
$\Gamma_{11}, \mathfrak{t}_9$	$\mathfrak{t}_\mathbf{0} + 130$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 90$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 30$
$\Gamma_{12}, \mathfrak{t}_{10}$	$\mathfrak{t}_\mathbf{0} + 135$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 95$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 30$
$\Gamma_{13}, \mathfrak{t}_{11}$	$\mathfrak{t}_\mathbf{0} + 145$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 105$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 30$
$\Gamma_{14}, \mathfrak{t}_{12}$	$\mathfrak{t}_\mathbf{0} + 150$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 110$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 30$
$\Gamma_{15}, \mathfrak{t}_{13}$	$\mathfrak{t}_\mathbf{0} + 1140$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 740$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 400$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 40) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 300$
$\Gamma_{16}, \mathfrak{t}_{14}$	$\mathfrak{t}_\mathbf{0} + 1540$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 740$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 400$	1	$\mathfrak{t}_\mathbf{0} + 1140$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 1140) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 700$
$\Gamma_{17}, \mathfrak{t}_{15}$	$\mathfrak{t}_\mathbf{0} + 1640$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 740$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 400$	1	$\mathfrak{t}_\mathbf{0} + 1140$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 1140) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 800$
$\Gamma_{18}, \mathfrak{t}_{16}$	$\mathfrak{t}_\mathbf{0} + 1840$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 740$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s} + 40) \cdot 3 + 400$	0	$\mathfrak{t}_\mathbf{0} + 1640$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 1140) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 102000$
$\Gamma_{19}, \mathfrak{t}_{17}$	$\mathfrak{t}_\mathbf{0} + 184 \cdot N$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 74 \cdot N$	$\tau_\mathbf{0}^\mathbf{s}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 52 \cdot N$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 114 \cdot N) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 10200 \cdot N$

Table 6. Global states for the Buffering algorithm.

consumption is added to its energy usage. The time-dependent energy usage in this case is $500 \cdot (2 + 200 \cdot 1) = 101000$.

We now need to overestimate the result of the outer loop. We add the difference of states Γ_{18} and Γ_1, multiplied by the loop bound $\lceil N/10 \rceil$. Notice that we do not know whether the radio is on upon entering the loop. After one or more iteration, the radio is turned off. The (aWhile) rule first calculates the maximal state in which the loop may actually be entered (the worst-case iteration). For $C_r :: s.on$ this gives the result $\max(0, \mathbf{on}_\mathbf{0}^\mathbf{r}) = \mathbf{on}_\mathbf{0}^\mathbf{r}$. This gives an overestimation only if $\mathbf{on}_\mathbf{0}^\mathbf{r} = 1$. Moreover, after the loop $C_r :: s.on = \mathbf{on}_\mathbf{0}^\mathbf{r}$.

After applying the Hoare logic, we still need to add the time-dependent energy-consumption for each component. The implicit component does not consume energy based on time. Its energy consumption at Γ_{19} is thus final. The result for the sensor is $\mathfrak{e}_\mathbf{0}^\mathbf{s} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 + 52 \cdot N + (\mathfrak{t}_\mathbf{0} + 184 \cdot N - \tau_\mathbf{0}^\mathbf{s}) \cdot 3 = \mathfrak{e}_\mathbf{0}^\mathbf{s} + 6 \cdot \mathfrak{t}_\mathbf{0} - 3 \cdot \tau_\mathbf{0}^\mathbf{s} + 604 \cdot N$. The result for the radio is $\mathfrak{e}_\mathbf{0}^\mathbf{r} + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r} + 114 \cdot N) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 10200 \cdot N + (\mathfrak{t}_\mathbf{0} + 184 \cdot N - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) = \mathfrak{e}_\mathbf{0}^\mathbf{r} + (2 \cdot (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) + 298 \cdot N) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 10200 \cdot N$.

Finally, we can calculate the sum of the energy-usage of all components:

$$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 74 \cdot N + \mathfrak{e}_\mathbf{0}^\mathbf{s} + 6 \cdot \mathfrak{t}_\mathbf{0} - 3 \cdot \tau_\mathbf{0}^\mathbf{s} + 604 \cdot N + \mathfrak{e}_\mathbf{0}^\mathbf{r} + (2 \cdot (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) + 298 \cdot N) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 10200 \cdot N =$$

$$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + \mathfrak{e}_\mathbf{0}^\mathbf{s} + \mathfrak{e}_\mathbf{0}^\mathbf{r} + 6 \cdot \mathfrak{t}_\mathbf{0} - 3 \cdot \tau_\mathbf{0}^\mathbf{s} + 10878 \cdot N + 2 \cdot (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot 2 + 2 \cdot (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r} + 298 \cdot N \cdot 2 + 298 \cdot N \cdot 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r} =$$

$$\mathfrak{e}_0^{\mathbf{cpu}} + \mathfrak{e}_0^{\mathbf{s}} + \mathfrak{e}_0^{\mathbf{r}} + 6 \cdot \mathfrak{t}_0 - 3 \cdot \tau_0^{\mathbf{s}} + 11474 \cdot N + (4 + 400 \cdot \mathbf{on}_0^{\mathbf{r}}) \cdot (\mathfrak{t}_0 - \tau_0^{\mathbf{r}}) + 59600 \cdot N \cdot \mathbf{on}_0^{\mathbf{r}}$$

This constitutes an energy usage that is symbolic over both the input variables of the program and starting state of the components. If we assume an "empty" starting state, in which the radio has just been turned off and no energy has been consumed yet, then this results in:

$$11474 \cdot N$$

5 Soundness of the Logic with Respect to the Semantics

Soundness is provided if the previous annotations (loop bounds and symbolic states) are sound. With wrong annotation, soundness is not provided). In the following we will assume the annotations to be correct.

Theorem 1 (Time dependent function properties.). *The following properties holds for time dependent functions.*

- *Let* $td(C_i, \mathfrak{t}_1) = \mathfrak{e}_1$ *and* $td(C_i, \mathfrak{t}_2) = \mathfrak{e}_2$*, if* $\mathfrak{t}_1 \geq \mathfrak{t}_2$ *then* $\mathfrak{e}_1 \geq \mathfrak{e}_2$*.*
- *Let* $td(C_i^{\Gamma_1}, \mathfrak{t}) = \mathfrak{e}_1$ *and* $td(C_i^{\Gamma_2}, \mathfrak{t}) = \mathfrak{e}_2$*, if* $C_i^{\Gamma_1} :: s \geq C_i^{\Gamma_2} :: s_2$ *and* $C_i^{\Gamma_1} :: \mathfrak{t} \leq C_i^{\Gamma_2} :: \mathfrak{t}$ *then* $\mathfrak{e}_1 \geq \mathfrak{e}_2$

Proof. First property follows directly from its definition. Indeed, if the timestamp is greater, then the difference between $(\mathfrak{t} - C_i :: \tau)$ is greater.

Also the second property follows from its definition since if the state is greater, then the function $C_i :: \phi(C_i :: s)$ return a greater value or if the internal timestamp is smaller, than the difference $(\mathfrak{t} - C_i :: \tau)$ is greater; hence the property holds.

Lemma 1 (Transitive property). *We show that* $\geq$ *satisfy the transitive property. If* $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma_2; \mathfrak{t}_2)$ *and* $(\Gamma_2; \mathfrak{t}_2) \geq (\Gamma_3; \mathfrak{t}_3)$*, then* $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma_3; \mathfrak{t}_3)$*.*

Proof. Notice that if $\forall i$, $td(C_i, \mathfrak{t}_1) + C_i^{\Gamma_1} :: \mathfrak{e} \geq td(C_i, \mathfrak{t}_2) + C_i^{\Gamma_2} :: \mathfrak{e}$ and $\forall i$, $td(C_i, \mathfrak{t}_2) + C_i^{\Gamma_1} :: \mathfrak{e} \geq td(C_i, \mathfrak{t}_3) + C_i^{\Gamma_3} :: \mathfrak{e}$ then surely thesis follows.

This concludes the proof. □

Lemma 2. *Let* $\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S \{\Gamma_2; \mathfrak{t}_2; \rho_2\}$ *and* $\{\Gamma_3; \mathfrak{t}_3; \rho_3\} S \{\Gamma_4; \mathfrak{t}_4; \rho_4\}$*; For every component* C_i*, if* $C_i^{\Gamma_1} :: s \geq C_i^{\Gamma_3} :: s$ *then* $C_i^{\Gamma_2} :: s \geq C_i^{\Gamma_4} :: s$

Proof. By structural induction on the derivation tree.

- If last rule was $(aSkip), (aConst), (aVar)$, thesis follows directly.
- If last rule was $(aBinOp), (aAssign)$, thesis follows by induction on the premises. No state is changed in the rule.
- If last rule applied was $(aCallCmpF)$, then thesis follows by Axiom 2.
- If last rule was $(aCallF)$, thesis follows by induction on the hypothesis.
- If last rule was $(aConcat)$ then we have the following two derivation trees:

$$\frac{\dfrac{\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S_1 \{\Gamma_2; \mathfrak{t}_2; \rho_2\} \quad \{\Gamma_2; \mathfrak{t}_2; \rho_2\} S_2 \{\Gamma_3; \mathfrak{t}_3; \rho_3\}}{\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S_1; S_2 \{\Gamma_3 \mathfrak{t}_3; \rho_3\}} (\text{aConcat}) \quad \dfrac{\{\Gamma_4; \mathfrak{t}_4; \rho_4\} S_1 \{\Gamma_5; \mathfrak{t}_5; \rho_5\} \quad \{\Gamma_5; \mathfrak{t}_5; \rho_5\} S_2 \{\Gamma_6; \mathfrak{t}_6; \rho_6\}}{\{\Gamma_4; \mathfrak{t}_4; \rho_4\} S_1; S_2 \{\Gamma_6 \mathfrak{t}_6; \rho_6\}} (\text{aConcat})}{}$$

  We apply induction hypothesis on the first premises $\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S_1 \{\Gamma_2; \mathfrak{t}_2; \rho_2\}$ and $\{\Gamma_4; \mathfrak{t}_4; \rho_4\} S_1 \{\Gamma_5; \mathfrak{t}_5; \rho_5\}$. This assure us to apply induction hypothesis also on the second premises $\{\Gamma_2; \mathfrak{t}_2; \rho_2\} S_2 \{\Gamma_3; \mathfrak{t}_3; \rho_3\}$ that is on the first derivation and $\{\Gamma_5; \mathfrak{t}_5; \rho_5\} S_2 \{\Gamma_6; \mathfrak{t}_6; \rho_6\}$ that appears in the second derivation. The result of applying induction hypothesis prove this case, since $\{\Gamma_3; \mathfrak{t}_3; \rho_3\}$ and $\{\Gamma_6; \mathfrak{t}_6; \rho_6\}$ are also the post-condition of the terms in the root of the derivation.
- If last rule was (aIf), then by applying induction hypothesis on both branches we get the thesis. Indeed, the function **lub**() assure us to retrieve the biggest states.

– The latter case is when the last rule was $(aWhile)$. We have the following two derivation trees

$$\frac{\Gamma_2 = \textbf{process-td}(\Gamma_1, \mathfrak{t}_1) \qquad \Gamma_4 = \Gamma_3[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \{\textbf{wci}(\Gamma_2, e; S); \mathfrak{t}_1; \rho\} e \{\Gamma_3; \mathfrak{t}_2; \rho_1\} \quad \{\Gamma_4; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_5; \mathfrak{t}_3; \rho_2\}}{\{\Gamma_1; \mathfrak{t}_1; \rho\} \textbf{while}_{ib}\ e\ \textbf{do}\ S\ \textbf{end while} \{\textbf{oe}(\Gamma_2, \mathfrak{t}_1, \Gamma_5, \mathfrak{t}_3, ib); \rho_3\}} (\text{aWhile})$$

$$\frac{\Gamma_7 = \textbf{process-td}(\Gamma_6, \mathfrak{t}_6) \qquad \Gamma_9 = \Gamma_8[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \{\textbf{wci}(\Gamma_7, e; S); \mathfrak{t}_6; \rho\} e \{\Gamma_8; \mathfrak{t}_7; \rho_1\} \quad \{\Gamma_9; \mathfrak{t}_7 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_{10}; \mathfrak{t}_8; \rho_2\}}{\{\Gamma_6; \mathfrak{t}_6; \rho\} \textbf{while}_{ib}\ e\ \textbf{do}\ S\ \textbf{end while} \{\textbf{oe}(\Gamma_7, \mathfrak{t}_6, \Gamma_{10}, \mathfrak{t}_8, ib); \rho_3\}} (\text{aWhile})$$

First, notice that if $\Gamma_1 \geq \Gamma_6$ then $\textbf{process-td}(\Gamma_1, \mathfrak{t}_1) \geq \textbf{process-td}(\Gamma_6, \mathfrak{t}_6)$ because state does not change. Moreover, $\textbf{wci}(\Gamma_2, e; S) \geq \textbf{wci}(\Gamma_7, e; S)$, since by its definition, it preserves the ordering. We can apply induction hypothesis on $\{\textbf{wci}(\Gamma_2, e; S); \mathfrak{t}_1; \rho\} e \{\Gamma_3; \mathfrak{t}_2; \rho_1\}$ and on $\{\textbf{wci}(\Gamma_7, e; S); \mathfrak{t}_6; \rho\} e \{\Gamma_8; \mathfrak{t}_7; \rho_1\}$; we get $\Gamma_8 \geq \Gamma_3$.
We can now apply induction hypothesis also on between $\{\Gamma_4; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_5; \mathfrak{t}_3; \rho_2\}$ and $\{\Gamma_9; \mathfrak{t}_7 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_{10}; \mathfrak{t}_8; \rho_2\}$.
By concatenating all the implications we have that for every i, if $C_i^{\Gamma_1} \geq C_i^{\Gamma_6}$ then $C_i^{\Gamma_5} \geq C_i^{\Gamma_{10}}$.
Now notice that the function **oe** do three things. Find the largest non-energy-aware output state, which still preserve the ordering, find the minimal timestamps, which does not influence the component state, and add the resource consumption of the loop itself which don't influence the final state. Finally it performs a **lub** between input state and output state. Since $\Gamma_1 \geq \Gamma_6$ and $\Gamma_5 \geq \Gamma_{10}$ the thesis is proven.

This concludes the proof. □

Corollary 1. *Analysis preserve the ordering on the states.*

Proof. by theorem 2

In the following $\Gamma^{\mathfrak{e}+n}$ represents a Component State Γ_1 equal to Γ except that the sum of all the energy consumed in every $C_i^{\Gamma_1}$ is equal to the sum of energies consumed in Γ plus n.

Theorem 2 (Power already consumed). *The analysis doesn't depend on the original energy already consumed. Be $n > 0$, if $\{\Gamma; \mathfrak{t}; \rho\} S_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ then $\{\Gamma^{\mathfrak{e}+n}; \mathfrak{t}; \rho\} S_1 \{\Gamma_1^{\mathfrak{e}+n}; \mathfrak{t}_1); \rho_1\}$*

Proof. By structural induction on the production derivation tree and on the premises. □

Lemma 3 (Timestamp). *The analysis depends on the original timing. Bigger timing on input gives bigger timing and power consumption on output. Be $n \geq 0$, if $\{\Gamma; \mathfrak{t}; \rho\} S_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ then $\{\Gamma; \mathfrak{t} + n; \rho\} S_1 \{\Gamma_1'; \mathfrak{t}_1 + n; \rho_1\}$, where the sum of energy consumed in Γ_1' is greater than the one in Γ_1 (Γ_1' differs from Γ_1 just on the energy consumed).*

Proof. By structural induction on the production derivation tree and on the premises. □

Theorem 3 (Analysis always overestimate the same time).
If $\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S \{\Gamma_2; \mathfrak{t}_2; \rho_2\}$ and $\{\Gamma_3; \mathfrak{t}_3; \rho_3\} S \{\Gamma_4; \mathfrak{t}_4; \rho_4\}$ then $\mathfrak{t}_2 - \mathfrak{t}_1 = \mathfrak{t}_4 - \mathfrak{t}_3$.

Proof. By structural induction on the production derivation tree and on the premises.

Theorem 4 (Timestamp analysis). *If $\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S \{\Gamma_2; \mathfrak{t}_2; \rho_2\}$, for all pair of state $\Gamma'; \mathfrak{t}'$ such that we can have the following derivation $\Delta \vdash \langle S, \sigma', \Gamma', \mathfrak{t}' \rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}'' \rangle$, then we have $\mathfrak{t}_2 - \mathfrak{t}_1 \geq \mathfrak{t}'' - \mathfrak{t}'$.*

Proof. By structural induction on the production derivation tree for S.

– If last rule was $(aConst)$ or $(aVar)$ or $(aSkip)$ then thesis holds.
– If last rule was $(aBinOp)$, then we have these two derivations:

$$\frac{\{\Gamma; \mathfrak{t}; \rho\} e_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\} \quad \{\Gamma_1; \mathfrak{t}_1; \rho_1\} e_2 \{\Gamma_2; \mathfrak{t}_2; \rho_2\} \quad \Gamma_3 = \Gamma_2[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma; \mathfrak{t}; \rho\} e_1 \boxdot e_2 \{\Gamma_3; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_e; \rho_2\}} (\text{aBinOp})$$

$$\frac{\dfrac{\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}' \rangle \qquad C_{imp} :: \boxdot(e_1, e_2) = p \quad \Delta \vdash \langle e_2, \sigma', \Gamma', \mathfrak{t}' \rangle \Downarrow^e \langle m, \sigma'', \Gamma'', \mathfrak{t}'' \rangle \quad \Gamma''' = \Gamma''[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\Delta \vdash \langle e_1 \boxdot e_2, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle p, \sigma'', \Gamma''', \mathfrak{t}'' + C_{imp} :: \mathfrak{T}_e \rangle}}{\Delta \vdash \langle e_1 \boxdot e_2, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' + C_{imp} :: \mathfrak{T}_e \rangle} (\text{eBinOp})$$

By applying induction hypothesis we easily retrieve the thesis, indeed same time is added on both derivations.

- If last rule was $(aAssign)$, as in the last case, by applying induction hypothesis we get the thesis.
- If last rule was $(aCallCmpF)$, property holds, since we are adding the same amount of time on both derivations.
- If last rule was $(aCallF)$, then by applying induction on the premise thesis follows.
- If last rule was $(aConcat)$, thesis follows by applying induction hypothesis on the premises.
- If last rule was (aIf), thesis follows by applying induction hypothesis on the correspondent premise and by noticing that the function max take the maximum between two timestamps.
- The most interesting case is when the last rules was $(aWhile)$. Concerning the semantics derivation, we could have applied $(eWhile - False)$ or $(eWhile - True)$. Clearly the interesting case is when the last semantic rule was $(eWhile - True)$. We have the following two derivations:

$$\dfrac{\dfrac{\dfrac{\Gamma_2 = \textbf{process-td}(\Gamma_1, \mathfrak{t}_1) \qquad \Gamma_4 = \Gamma_3[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \{\textbf{wci}(\Gamma_2, e; S); \mathfrak{t}_1; \rho\} e \{\Gamma_3; \mathfrak{t}_2; \rho_1\} \quad \{\Gamma_4; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_5; \mathfrak{t}_4; \rho_2\}}{\{\Gamma_1; \mathfrak{t}_1; \rho\} \textbf{while}_{ib}\ e\ \textbf{do}\ S\ \textbf{end while} \{\textbf{oe}(\Gamma_2, \mathfrak{t}_1, \Gamma_5, \mathfrak{t}_3, ib); \rho_3\}}(\text{aWhile}) \quad \Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \Delta \vdash \langle S_1, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w \rangle \Downarrow^s \langle \sigma''', \Gamma'''', \mathfrak{t}'''' \rangle \quad \Delta \vdash \langle \textbf{while}_{ib}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma''', \Gamma'''', \mathfrak{t}'''' \rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' \rangle}{\Delta \vdash \langle S_1; \textbf{while}_{ib}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w \rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' \rangle \quad n \neq 0}(\text{eConcat}) \quad \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}' \rangle}{\Delta \vdash \langle \textbf{while}_{ib}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' \rangle}(\text{eWhile-True})$$

We apply induction hypothesis on $\{\textbf{wci}(\Gamma_2, e; S); \mathfrak{t}_1; \rho\} e \{\Gamma_3; \mathfrak{t}_2; \rho_1\}$ and on $\langle e, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}' \rangle$. We have that $\mathfrak{t}_2 - \mathfrak{t}_1 \geq \mathfrak{t}' - \mathfrak{t}$. Similarly, we apply the induction hypothesis on $\{\Gamma_4; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_5; \mathfrak{t}_4; \rho_2\}$ and on $\Delta \vdash \langle S_1, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w \rangle \Downarrow^s \langle \sigma''', \Gamma'''', \mathfrak{t}'''' \rangle$. Clearly we may apply the induction hypothesis on all the sub-trees of $\Delta \vdash \langle \textbf{while}_{ib}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma''', \Gamma'''', \mathfrak{t}'''' \rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' \rangle$ in the semantic tree. Indeed, the induction says that this holds for every states $\Gamma; \mathfrak{t}$, which means that $\mathfrak{t}_4 - \mathfrak{t}_1$ is a good overestimation for every single possible cycles of the loop. Since ib is an overestimation of the loop, it follows that clerly $\mathfrak{t}_1 + (\mathfrak{t}_4 - \mathfrak{t}_1) * ib \geq \mathfrak{t}'' - \mathfrak{t}$ This concludes the proof. □

Corollary 2. *Analysis overestimates timestams.*

Proof. by theorem 4

The behaviour of the analysis is uniform. Whenever we start from an overestimated pair of states and timing, we finish with an overestimation of the result. Ordering is preserved.

Lemma 4 (Analysis preserve the ordering). *Let* $\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S \{\Gamma_2; \mathfrak{t}_2; \rho_2\}$ *and* $\{\Gamma_3; \mathfrak{t}_3; \rho_3\} S \{\Gamma_4; \mathfrak{t}_4; \rho_4\}$*; if* $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma_3; \mathfrak{t}_3)$ *then* $(\Gamma_2; \mathfrak{t}_2) \geq (\Gamma_4; \mathfrak{t}_4)$

Proof. By structural induction on the derivation tree

- If last rule was $(aSkip), (aConst), (aVar)$, thesis follows directly. Nothing changes
- If last rule was $(aBinOp), (aAssign)$, thesis follows by induction on the premises and because of Axiom 2 and because we add the same timestamp on both derivation.
- If last rule applied was $(aCallCmpF)$, then thesis follows because of Axiom 2 and because we are adding the same amount of energy and the same amount of time.
- If last rule was $(aCallF)$, thesis follows by induction on the hypothesis.
- If last rule was $(aConcat)$ then we have the following two derivation trees:

$$\dfrac{\dfrac{\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S_1 \{\Gamma_2; \mathfrak{t}_2; \rho_2\} \quad \{\Gamma_2; \mathfrak{t}_2; \rho_2\} S_2 \{\Gamma_3; \mathfrak{t}_3; \rho_3\}}{\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S_1; S_2 \{\Gamma_3 \mathfrak{t}_3; \rho_3\}}(\text{aConcat}) \quad \dfrac{\{\Gamma_4; \mathfrak{t}_4; \rho_4\} S_1 \{\Gamma_5; \mathfrak{t}_5; \rho_5\} \quad \{\Gamma_5; \mathfrak{t}_5; \rho_5\} S_2 \{\Gamma_6; \mathfrak{t}_6; \rho_6\}}{\{\Gamma_4; \mathfrak{t}_4; \rho_4\} S_1; S_2 \{\Gamma_6 \mathfrak{t}_6; \rho_6\}}(\text{aConcat})}$$

  We apply induction hypothesis on the first premises $\{\Gamma_1; \mathfrak{t}_1; \rho_1\} S_1 \{\Gamma_2; \mathfrak{t}_2; \rho_2\}$ and $\{\Gamma_4; \mathfrak{t}_4; \rho_4\} S_1 \{\Gamma_5; \mathfrak{t}_5; \rho_5\}$. This assure us to apply induction hypothesis also on the second premises $\{\Gamma_2; \mathfrak{t}_2; \rho_2\} S_2 \{\Gamma_3; \mathfrak{t}_3; \rho_3\}$ and $\{\Gamma_5; \mathfrak{t}_5; \rho_5\} S_2 \{\Gamma_6; \mathfrak{t}_6; \rho_6\}$. The result of applying induction hypothesis prove this case, since $\{\Gamma_3; \mathfrak{t}_3; \rho_3\}$ and $\{\Gamma_6; \mathfrak{t}_6; \rho_6\}$ are also the post-conditions of the terms in the root of the derivation.
- If last rule was (aIf), then by applying induction hypothesis on both branches we get the thesis. Indeed, the function **lub**() assure us to retrieve the biggest component states and max assure us to take the biggest global timestamp. Thesis follows by Axiom 1

– the latest state is when we consider the $(aWhile)$. We have the following two derivation trees

$$\frac{\Gamma_2 = \textbf{process-td}(\Gamma_1, \mathfrak{t}_1) \qquad \Gamma_4 = \Gamma_3[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \{\textbf{wci}(\Gamma_2, e; S); \mathfrak{t}_1; \rho\} e \{\Gamma_3; \mathfrak{t}_2; \rho_1\} \quad \{\Gamma_4; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_5; \mathfrak{t}_4; \rho_2\}}{\{\Gamma_1; \mathfrak{t}_1; \rho\} \textbf{while}_{ib}\ e\ \textbf{do}\ S\ \textbf{end while}\{\textbf{oe}(\Gamma_2, \mathfrak{t}_1, \Gamma_5, \mathfrak{t}_3, ib); \rho_3\}} (\text{aWhile})$$

$$\frac{\Gamma_7 = \textbf{process-td}(\Gamma_6, \mathfrak{t}_6) \qquad \Gamma_9 = \Gamma_8[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \{\textbf{wci}(\Gamma_7, e; S); \mathfrak{t}_6; \rho\} e \{\Gamma_8; \mathfrak{t}_7; \rho_1\} \quad \{\Gamma_9; \mathfrak{t}_7 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_{10}; \mathfrak{t}_9; \rho_2\}}{\{\Gamma_6; \mathfrak{t}_6; \rho\} \textbf{while}_{ib}\ e\ \textbf{do}\ S\ \textbf{end while}\{\textbf{oe}(\Gamma_7, \mathfrak{t}_6, \Gamma_{10}, \mathfrak{t}_8, ib); \rho_3\}} (\text{aWhile})$$

We assume, so, that $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma_6; \mathfrak{t}_6)$. Notice that reset function preserve the ordering by its definition, since it adds the power consumed in the slice of time between the previous timestamp and the present one. Hence $(\Gamma_2; \mathfrak{t}_1) \geq (\Gamma_7; \mathfrak{t}_6)$. Notice also that worst case iteration preserve the ordering since by its definition the fixpoint is reached with a concatenation of δ function (refer to Axiom 2). Hence $(\textbf{wci}(\Gamma_2, e; S); \mathfrak{t}_1) \geq (\textbf{wci}(\Gamma_7, e; S); \mathfrak{t}_6)$. This assure us that we can apply induction hypothesis on $\{\textbf{wci}(\Gamma_2, e; S); \mathfrak{t}_1; \rho\} e \{\Gamma_3; \mathfrak{t}_2; \rho_1\}$ and on $\{\textbf{wci}(\Gamma_7, e; S); \mathfrak{t}_6; \rho\} e \{\Gamma_8; \mathfrak{t}_7; \rho_1\}$ and get that $(\Gamma_3; \mathfrak{t}_2) \geq (\Gamma_8; \mathfrak{t}_7)$. Notice that resource function for the while actually doesn't change the components states, since is a resource function of the CPU. We add the same amount of energy on both derivation. Hence $(\Gamma_9; \mathfrak{t}_7 + C_{imp} :: \mathfrak{T}_w; \rho_2) \geq (\Gamma_4; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_w; \rho_2)$.
This means that we can apply induction hypothesis between $\{\Gamma_4; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_5; \mathfrak{t}_4; \rho_2\}$ and $\{\Gamma_9; \mathfrak{t}_7 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_{10}; \mathfrak{t}_9; \rho_2\}$. By concatenating the implications we have that $(\Gamma_5; \mathfrak{t}_3) \geq (\Gamma_{10}; \mathfrak{t}_8)$.
Now we have to apply the function **oe**; it does three things. It finds the largest non-energy-aware output state. This preserves the ordering. It finds the minimal timestamps, which preserves the ordering of the pair, since we are backdating timestamp for the same slice of time (Lemma 3). Finally it performs a **lub** between input state and output state. This clearly preserve the ordering, since we are taking the maximum.
Thesis is proven.

This concludes the proof □

.

The analysis overestimate the state of each component.

Theorem 5 (State overestimation). *The analysis overestimate the state of each component. The following two thesis holds:*

– *If* $\{\Gamma; \mathfrak{t}; \rho\} S \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ *and* $\Delta \vdash \langle S, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}' \rangle$ *then for every device* i, $C_i^{\Gamma_1} :: s \geq C_i^{\Gamma'} :: s$.
– *If* $\{\Gamma; \mathfrak{t}; \rho\} e \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ *and* $\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle e', \sigma', \Gamma', \mathfrak{t}' \rangle$ *then for every device* i, $C_i^{\Gamma_1} :: s \geq C_i^{\Gamma'} :: s$.

Proof. Proof is proven by induction on the premises and on the derivation tree. Clearly we need these two induction hypothesis at the same time, since an expression can be seen as a statement. Analysis doesn't change, while the semantics differs.

– If last rule in the analysis was $(aConst)$ or $(aVar)$, thesis holds since no component has been touched.
– If last rule in the analysis was $(aBinOp)$ we have these two derivations:

$$\frac{\{\Gamma; \mathfrak{t}; \rho\} e_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\} \quad \{\Gamma_1; \mathfrak{t}_1; \rho_1\} e_2 \{\Gamma_2; \mathfrak{t}_2; \rho_2\} \quad \Gamma_3 = \Gamma_2[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma; \mathfrak{t}; \rho\} e_1 \boxdot e_2 \{\Gamma_3; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_e; \rho_2\}} (\text{aBinOp})$$

$$\frac{\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}' \rangle \qquad C_{imp} :: \boxdot(e_1, e_2) = p \quad \Delta \vdash \langle e_2, \sigma', \Gamma', \mathfrak{t}' \rangle \Downarrow^e \langle m, \sigma'', \Gamma'', \mathfrak{t}'' \rangle \quad \Gamma''' = \Gamma''[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\Delta \vdash \langle e_1 \boxdot e_2, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle p, \sigma'', \Gamma''', \mathfrak{t}'' + C_{imp} :: \mathfrak{T}_e \rangle} (\text{eBinOp})$$

We can apply the induction hypothesis on $\{\Gamma; \mathfrak{t}; \rho\} e_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ and $\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}' \rangle$. Induction hypothesis assure us that for every i $C_i^{\Gamma_1} \geq C_i^{\Gamma'}$. Surely we cannot use no more the induction hypothesis, since the premises of the second derivations (the ones for e_2) differ.
We re-create a new analysis derivation just for the purpose of proving our thesis. We create the derivation $\{\Gamma'; \mathfrak{t}'; \rho'\} e_2 \{\Gamma_4; \mathfrak{t}_4; \rho_4\}$. Since we start from same premises, we can apply induction hypothesis with the derivation in the semantics tree $\Delta \vdash \langle e_2, \sigma', \Gamma', \mathfrak{t}' \rangle \Downarrow^e \langle m, \sigma'', \Gamma'', \mathfrak{t}'' \rangle$, obtaining that for every i, $C_i^{\Gamma_4} \geq C_i^{\Gamma''}$. Recall that for every i, $C_i^{\Gamma_1} \geq C_i^{\Gamma'}$. We can hence use the Lemma 2 and get the following chain: for every i, $C_i^{\Gamma_2} \geq C_i^{\Gamma_4} \geq C_i^{\Gamma''}$.
Observing that CPU is a stateless component and that we are adding the same amount of time on final timestamp, we can deduce that $C_i^{\Gamma_3} \geq C_i^{\Gamma'''}$.

- If last production rule was ($aAssign$), then the last semantic rule was ($eAssign$). Thesis is proven by induction on the premise.
- If last production rule was ($aCallCmpF$), then the last semantic rule was ($eCallCmpF$). We are applying the same $C_i :: \delta_f()$ on both side.
- If last production rule was ($aCallF$), then the last semantic rule was ($eCallF$). Thesis is proven by induction on the premise.
- If last semantic rule was ($eExprAsStmt$), then whatever is the production rule, thesis is proven by induction on the premise of the semantic tree.
- If last production rule was ($aSkip$), then the last semantic rule was ($eSkip$). Thesis holds.
- If last production rule was ($aConcat$), then the last semantic rule was ($eConcat$). We have these two derivation trees:

$$\frac{\{\Gamma;\mathfrak{t};\rho\}S_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \quad \{\Gamma_1;\mathfrak{t}_1;\rho_1\}S_2\{\Gamma_2;\mathfrak{t}_2;\rho_2\}}{\{\Gamma;\mathfrak{t};\rho\}S_1;S_2\{\Gamma_2;\mathfrak{t}_2;\rho_2\}}(\text{aConcat})$$

$$\frac{\Delta \vdash \langle S_1,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^s \langle\sigma',\Gamma',\mathfrak{t}'\rangle \quad \Delta \vdash \langle S_2,\sigma',\Gamma',\mathfrak{t}'\rangle \Downarrow^s \langle\sigma'',\Gamma'',\mathfrak{t}''\rangle}{\Delta \vdash \langle S_1;S_2,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^s \langle\sigma'',\Gamma'',\mathfrak{t}''\rangle}(\text{eConcat})$$

  We can cleary apply induction hypothesis on $\{\Gamma;\mathfrak{t};\rho\}S_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\}$ and $\Delta \vdash \langle S_1,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^s \langle\sigma',\Gamma',\mathfrak{t}'\rangle$ and get that for every i, $C_i^{\Gamma_1} \geq C_i^{\Gamma'}$.
  We now create a derivation $\{\Gamma';\mathfrak{t}';\rho'\}S_2\{\Gamma_3;\mathfrak{t}_3;\rho_3\}$ just for the purpose of proving the thesis. We can apply induction hypothesis with $\Delta \vdash \langle S_2,\sigma',\Gamma',\mathfrak{t}'\rangle \Downarrow^s \langle\sigma'',\Gamma',\mathfrak{t}'\rangle$ and get that for every i, $C_i^{\Gamma_3} \geq C_i^{\Gamma''}$. Since $\Gamma_1 \geq \Gamma'$ we can apply Lemma 2 and get the thesis $C_i^{\Gamma_2} \geq C_i^{\Gamma_3} \geq C_i^{\Gamma''}$.
- If last production rule was (aIf), then we could have two possible semantics ruled applied. We just show the case where the "then" branch is taken. The other case is similar to this one. We have these two derivations:

$$\frac{\begin{array}{cc}\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} & \{\Gamma_2;\mathfrak{t}_1 + C_{imp}::\mathfrak{T}_{ite};\rho_1\}S_1\{\Gamma_3;\mathfrak{t}_2;\rho_3\} \\ \Gamma_2 = \Gamma_1[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_{ite}] & \{\Gamma_2;\mathfrak{t}_1 + C_{imp}::\mathfrak{T}_{ite};\rho_1\}S_2\{\Gamma_4;\mathfrak{t}_3;\rho_4\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if}\{\textbf{lub}(\Gamma_3,\Gamma_4);\max\{\mathfrak{t}_2,\mathfrak{t}_3\};\rho_5\}}(\text{aIf})$$

$$\frac{\begin{array}{cc}\Delta \vdash \langle e,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle 0,\sigma',\Gamma',\mathfrak{t}'\rangle & \Delta \vdash \langle S_1,\sigma',\Gamma'',\mathfrak{t}' + C_{imp}::\mathfrak{T}_{ite}\rangle \Downarrow^s \langle\sigma'',\Gamma''',\mathfrak{t}''\rangle \\ & \Gamma'' = \Gamma'[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_{ite}]\end{array}}{\Delta \vdash \langle \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if},\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^s \langle\sigma'',\Gamma''',\mathfrak{t}''\rangle}(\text{eIf-False})$$

  We can apply induction hypothesis on $\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\}$ and $\Delta \vdash \langle e,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle 0,\sigma',\Gamma',\mathfrak{t}'\rangle$ and retrieve that for every i, $C_i^{\Gamma_1} \geq C_i^{\Gamma'}$. Since we are working with cpu, no state changes and, moreover, we are adding the same amount of time on the following premises on each derivation. we deduce that i, $C_i^{\Gamma_2} \geq C_i^{\Gamma''}$. Now we cannot apply induction hypothesis on the two branches for S_1.
  Therefore, we create a new production derivation $\{\Gamma'';\mathfrak{t}' + C_{imp}::\mathfrak{T}_{ite};\rho''\}S_1\{\Gamma_6;\mathfrak{t}_6;\rho_6\}$. We can apply induction hypothesis with the correspondent branch in the semantic tree and get that for every i, $C_i^{\Gamma_6} \geq C_i^{\Gamma'''}$. Notice also that $\Gamma_2 \geq \Gamma''$ and hence we can apply the Lemma 2 and retrieve that for every i, $C_i^{\Gamma_3} \geq C_i^{\Gamma_6} \geq C_i^{\Gamma'''}$. Clearly the $\textbf{lub}(\Gamma_3,\Gamma_4) \geq \Gamma_3$ and hence the thesis is proven.
- If last production rule was ($aWhile$), we could have two possible last semantic ruled applied. We consider the case where the last semantic rule was ($eWhile{-}True$); in the other case, thesis follows straightforward. We have the following derivation trees:

$$\frac{\begin{array}{cc}\Gamma_1 = \textbf{process-td}(\Gamma,\mathfrak{t}) & \Gamma_3 = \Gamma_2[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_w] \\ \{\textbf{wci}(\Gamma_1,e;S);\mathfrak{t};\rho\}e\{\Gamma_2;\mathfrak{t}_1;\rho_1\} & \{\Gamma_3;\mathfrak{t}_1 + C_{imp}::\mathfrak{T}_w;\rho_1\}S\{\Gamma_4;\mathfrak{t}_2;\rho_2\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\textbf{while}_{ib}\ e \textbf{ do } S \textbf{ end while}\{\textbf{oe}(\Gamma_1,\mathfrak{t},\Gamma_4,\mathfrak{t}_2,ib);\rho_3\}}(\text{aWhile})$$

$$\frac{\begin{array}{cc}\Gamma'' = \Gamma'[C_{imp}::\mathfrak{e} \mathrel{+}= C_{imp}::\mathfrak{E}_w] & \Delta \vdash \langle S_1,\sigma',\Gamma'',\mathfrak{t}' + C_{imp}::\mathfrak{T}_w\rangle \Downarrow^s \langle\sigma''',\Gamma'''',\mathfrak{t}'''\rangle \quad \Delta \vdash \langle \textbf{while}_{ib}\ e \textbf{ do } S_1 \textbf{ end while},\sigma''',\Gamma'''',\mathfrak{t}'''\rangle \Downarrow^s \langle\sigma'',\Gamma''',\mathfrak{t}''\rangle \\ \Delta \vdash \langle e,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle n,\sigma',\Gamma',\mathfrak{t}'\rangle & \Delta \vdash \langle S_1;\textbf{while}_{ib}\ e \textbf{ do } S_1 \textbf{ end while},\sigma',\Gamma'',\mathfrak{t}' + C_{imp}::\mathfrak{T}_w\rangle \Downarrow^s \langle\sigma'',\Gamma''',\mathfrak{t}''\rangle \quad n \neq 0\end{array}}{\Delta \vdash \langle \textbf{while}_{ib}\ e \textbf{ do } S_1 \textbf{ end while},\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^s \langle\sigma'',\Gamma''',\mathfrak{t}''\rangle}(\text{eWhile-True})^{\text{sC}}$$

  Notice that $\textbf{wci}(\Gamma,e;S) \geq \Gamma$ by its definition. Hence we can apply induction on a particular production derivation $\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma^1;\mathfrak{t}^1;\rho_1\}$ and $\Delta \vdash \langle e,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle n,\sigma',\Gamma',\mathfrak{t}'\rangle$. Then, by lemma 2, we get that for every i, $C_i^{\Gamma_1} \geq C_i^{\Gamma'}$.
  In the next Hoare sub-tree no state is changed and hence ordering is preserved; Just for the purpose of proving the theorem we create another different derivation on which we can apply induction. We apply induction hypothesis between $\{\Gamma'';\mathfrak{t}' + C_{imp}::\mathfrak{T}_w;\rho_1\}S\{\Gamma_3';\mathfrak{t}_2';\rho_2'\}$ and $\Delta \vdash \langle S_1,\sigma',\Gamma'',\mathfrak{t}' + C_{imp}::\mathfrak{T}_w\rangle \Downarrow^s \langle\sigma''',\Gamma'''',\mathfrak{t}'''\rangle$. Then, by lemma 2, we get that for every i, $C_i^{\Gamma_4} \geq C_i^{\Gamma''''}$.
  By definition of worst case iteration, notice that Γ_4 is greater than all the component states that could be found at the end of each cycle in $\Delta \vdash \langle S_1;\textbf{while}_{ib}\ e \textbf{ do } S_1 \textbf{ end while},\sigma',\Gamma''',\mathfrak{t}'''\rangle \Downarrow^s \langle\sigma'',\Gamma'''',\mathfrak{t}''''\rangle$.
  We have now to apply the function **oe**, which does three things.

It finds the largest non-energy-aware output state. This preserves the ordering. It finds the minimal timestamps, which does not influences the component states. Finally it performs a **lub** between input state and output state. This clearly preserve the ordering, since we are taking the maximum.
The final **lub** assure us that the final state found out with the analysis is greater than every possible state in the real calculation. Indeed, Γ_4 is bigger than all the component state that could be found at the end of every cycle. It doesn't say that is the biggest state respect to the input state (hence, in the case where we did not enter in the while loop). Final least upper-bound assure us this.

This concludes the proof. □

Theorem 6 (soundness). *For every global timing* $\mathfrak{t}$ *and for every set of component states* Γ*, if* $\Delta \vdash \langle S_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle$ *and* $\{\mathfrak{t}; \Gamma; \rho\} S_1 \{\mathfrak{t}_1; \Gamma_1; \rho_1\}$ *or if* $\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle \sigma', \Gamma', \mathfrak{t}'\rangle$ *and* $\{\mathfrak{t}; \Gamma; \rho\} S_1 \{\mathfrak{t}_1; \Gamma_1; \rho_1\}$ *, then it holds that* $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma'; \mathfrak{t}')$*. The final states overestimate the actual ones.*

Proof. On induction on the premises and on the derivation tree.

- If last semantic rule was ($eConst$) or ($eVar$), thesis holds, since nothing has been changed.
- If last semantic rule was ($eBinOp$), last production rule was surely ($aBinOp$). We have the following case:

$$\frac{\{\Gamma; \mathfrak{t}; \rho\} e_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\} \quad \{\Gamma_1; \mathfrak{t}_1; \rho_1\} e_2 \{\Gamma_2; \mathfrak{t}_2; \rho_2\} \quad \Gamma_3 = \Gamma_2[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]}{\{\Gamma; \mathfrak{t}; \rho\} e_1 \boxdot e_2 \{\Gamma_3; \mathfrak{t}_2 + C_{imp} :: \mathfrak{T}_e; \rho_2\}} (\text{aBinOp})$$

$$\frac{\begin{array}{c}\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \qquad C_{imp} :: \boxdot(e_1, e_2) = p \\ \Delta \vdash \langle e_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^e \langle m, \sigma'', \Gamma'', \mathfrak{t}''\rangle \quad \Gamma''' = \Gamma''[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_e]\end{array}}{\Delta \vdash \langle e_1 \boxdot e_2, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle p, \sigma'', \Gamma''', \mathfrak{t}'' + C_{imp} :: \mathfrak{T}_e\rangle} (\text{eBinOp})$$

  We apply induction hypothesis on $\{\Gamma; \mathfrak{t}; \rho\} e_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ and $\Delta \vdash \langle e_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle$ and get an overestimation on the result: $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma'; \mathfrak{t}')$. Just for the purpose of the analysis we create a new analysis $\{\Gamma'; \mathfrak{t}'; \rho_1\} e_2 \{\Gamma_2'; \mathfrak{t}_2'; \rho_2\}$ and apply induction hypothesis with $\Delta \vdash \langle e_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^e \langle m, \sigma'', \Gamma'', \mathfrak{t}''\rangle$. We get $(\Gamma_2'; \mathfrak{t}_2') \geq (\Gamma''; \mathfrak{t}'')$, but from lemma 4 we get that $(\Gamma_2; \mathfrak{t}_2) \geq (\Gamma''; \mathfrak{t}'')$. Hence, since at the end we are adding the same amount of time and energy on both derivation, the property still holds.
- If last semantic rule was ($eAssign$), then the last production rule was surely ($aAssign$). We have the following case:

$$\frac{\{\Gamma; \mathfrak{t}; \rho\} e \{\Gamma_1; \mathfrak{t}_1; \rho_1\} \quad \Gamma_2 = \Gamma_1[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\{\Gamma; \mathfrak{t}; \rho\} x = e \{\Gamma_2; \mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_a; \rho_2\}} (\text{aAssign})$$

$$\frac{\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_a]}{\Delta \vdash \langle x_1 = e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma[x_1 \leftarrow n], \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_a\rangle} (\text{eAssign})$$

  Clearly thesis holds by induction and by considering that we are adding the same amount of energy and time on both derivations.
- If last semantic rule was ($eCallCmpF$), then we have ($aCallCmpF$) as production rule; we have the following case:

$$\frac{\Gamma_1 = \Gamma[C_i :: s \leftarrow C_i :: \delta_f(C_i :: s), C_i :: \tau \leftarrow \mathfrak{t}, C_i :: \mathfrak{e} \mathrel{+}= C_i :: \mathfrak{E}_f + td(C_i, \mathfrak{t})]}{\{\Gamma; \mathfrak{t}; \rho\} C_i :: f(args) \{\Gamma_1; \mathfrak{t} + C_i :: \mathfrak{T}_f; \rho\}} (\text{aCallCmpF})$$

$$\frac{\begin{array}{c}\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle a, \sigma', \Gamma', \mathfrak{t}'\rangle \quad C_i :: rv_f(C_i^{\Gamma'} :: s, a) = n \\ \Gamma'' = \Gamma[C_i :: \mathfrak{e} \mathrel{+}= C_i :: \mathfrak{E}_f + td(C_i^{\Gamma'}, \mathfrak{t}), C_i :: s \leftarrow C_i :: \delta_f(C_i^{\Gamma'} :: s, a), C_i :: \tau \leftarrow \mathfrak{t}']\end{array}}{\Delta \vdash \langle C_i :: f(e), \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma, \Gamma'', \mathfrak{t}' + C_i :: \mathfrak{T}_f\rangle} (\text{eCallCmpF})$$

  Thesis holds, since the result is equal. We are modifying the argument in the same way. We add the same amount of energy and time at the end.
- If last semantic rule was ($eCallF$), clearly last production rule was (FL). We have the following case:

$$\frac{\begin{array}{cc}\Delta(f) = (e_1, x) & \{\Gamma; \mathfrak{t}; \rho\} e \{\Gamma_1; \mathfrak{t}_1; \rho_1\} \\ e = a \in \rho & \{\Gamma_1; \mathfrak{t}_1; \rho_1[x' \leftarrow a]\} e_1[x \leftarrow x'] \{\Gamma_2; \mathfrak{t}_2; \rho_2\} \quad x' \text{ fresh in } e_1\end{array}}{\{\Gamma; \mathfrak{t}; \rho\} f(e) \{\Gamma_2; \mathfrak{t}_2; \rho_2\}} (\text{aCallF})$$

$$\frac{\Delta(f) = (e_1, \Delta', x) \quad \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle a, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Delta' \vdash \langle e_1, [x \leftarrow a], \Gamma', \mathfrak{t}'\rangle \Downarrow^e \langle n, \sigma'', \Gamma'', \mathfrak{t}''\rangle}{\Delta \vdash \langle f(e), \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma'', \mathfrak{t}''\rangle}(\text{eCallF})$$

Thesis holds by induction on the premise. Particular attention should have been made on the substitutions. In the semantic rules we are substituting on the state of the program the value a which expresses the dependencies of the variable x. This is equal to substitute in all e_1 the value a to value x (indeed, x is the parameter!). On the analysis rules we actual substitute the parameter with a fresh variable which has a dependence. the new statements are not equal but it can be proven that are equal up to some idempotent substitutions.

- If last semantic rule was ($eExprAsStmt$), then there are varius last production rules that could have been applied. All the cases can be brought back to all the case we have already analysed.
- If last semantic rule was ($eSkip$), then last production rule was ($aSkip$). Thesis holds, since nothing has been changed.
- If last semantic rule was ($eConcat$), then last production rule was ($aSkip$). We have the following case:

$$\frac{\{\Gamma; \mathfrak{t}; \rho\} S_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\} \quad \{\Gamma_1; \mathfrak{t}_1; \rho_1\} S_2 \{\Gamma_2; \mathfrak{t}_2; \rho_2\}}{\{\Gamma; \mathfrak{t}; \rho\} S_1; S_2 \{\Gamma_2; \mathfrak{t}_2; \rho_2\}}(\text{aConcat})$$

$$\frac{\Delta \vdash \langle S_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Delta \vdash \langle S_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}''\rangle}{\Delta \vdash \langle S_1; S_2, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}''\rangle}(\text{eConcat})$$

We apply induction hypothesis on $\{\Gamma; \mathfrak{t}; \rho\} S_1 \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ and $\Delta \vdash \langle S_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle$. We obtain that $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma'; \mathfrak{t}')$. Just for purpose of proof we create the new analysis $\{\Gamma'; \mathfrak{t}'; \rho_1\} S_2 \{\Gamma_2'; \mathfrak{t}_2'; \rho_2\}$ and we apply induction hypothesis with $\Delta \vdash \langle S_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^s \langle \sigma'', \Gamma'', \mathfrak{t}''\rangle$. We obtain that $(\Gamma_2'; \mathfrak{t}_2') \geq (\Gamma''; \mathfrak{t}'')$. By lemma 4 we conclude.

- If last semantic rule was ($eIf - False$), then last production rule was (aIf). We have the following case:

$$\frac{\begin{array}{cc} \{\Gamma; \mathfrak{t}; \rho\} e \{\Gamma_1; \mathfrak{t}_1; \rho_1\} & \{\Gamma_2; \mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_{ite}; \rho_1\} S_1 \{\Gamma_3; \mathfrak{t}_2; \rho_3\} \\ \Gamma_2 = \Gamma_1[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_{ite}] & \{\Gamma_2; \mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_{ite}; \rho_1\} S_2 \{\Gamma_4; \mathfrak{t}_3; \rho_4\} \end{array}}{\{\Gamma; \mathfrak{t}; \rho\} \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if} \{\textbf{lub}(\Gamma_3, \Gamma_4); \max\{\mathfrak{t}_2, \mathfrak{t}_3\}; \rho_5\}}(\text{aIf})$$

$$\frac{\begin{array}{cc} \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle 0, \sigma', \Gamma', \mathfrak{t}'\rangle & \Delta \vdash \langle S_1, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_{ite}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle \\ & \Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_{ite}] \end{array}}{\Delta \vdash \langle \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle}(\text{eIf-False})$$

Clearly we can apply induction hypothesis on $\{\Gamma; \mathfrak{t}; \rho\} e \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ and $\Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle 0, \sigma', \Gamma', \mathfrak{t}'\rangle$. We retrieve that $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma'; \mathfrak{t}')$. Since we are preserving the states and just adding the same energy consumption and timing on both derivation, we derive that $(\Gamma_2; \mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_{ite}; \rho_1) \geq (\Gamma''; \mathfrak{t}' + C_{imp} :: \mathfrak{T}_{ite})$. Just for the purpose of proof, we derive the following new analysis $\{\Gamma''; \mathfrak{t}' + C_{imp} :: \mathfrak{T}_{ite}; \rho_1\} S_1 \{\Gamma_3'; \mathfrak{t}_3'; \rho_3\}$ and we apply the induction hypothesis with $\Delta \vdash \langle S_1, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_{ite}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle$. We retrieve that $(\Gamma_3'; \mathfrak{t}_3') \geq (\Gamma'''; \mathfrak{t}'')$. By lemma 4 we conclude that $(\Gamma_3; \mathfrak{t}_3) \geq (\Gamma'''; \mathfrak{t}'')$.
We have to check if the result preserve the ordering. Notice the following chain: Clearly the result in the production derivation overestimates the actual result, since we are taking a (possible) greater state and a (possible) greater timestamp. Hence the case is valid.

- If last semantic rule was ($eIf - True$). This case is similar to the last case, where we analysed ($eIf - False$).
- If last semantic rule was ($eWhile - False$). Clearly last production rule was ($aWhile$). Thesis surely holds, since in the semantic rule nothing has been touched.
- If last semantic rule was ($eWhile - True$). Last production rule was ($aWhile$). We have the following case:

$$\frac{\begin{array}{cc} \Gamma_1 = \textbf{process-td}(\Gamma, \mathfrak{t}) & \Gamma_3 = \Gamma_2[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \\ \{\textbf{wci}(\Gamma_1, e; S); \mathfrak{t}; \rho\} e \{\Gamma_2; \mathfrak{t}_1; \rho_1\} & \{\Gamma_3; \mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_4; \mathfrak{t}_2; \rho_2\} \end{array}}{\{\Gamma; \mathfrak{t}; \rho\} \textbf{while}_{ib} \ e \ \textbf{do} \ S \ \textbf{end while} \{\textbf{oe}(\Gamma_1, \mathfrak{t}, \Gamma_4, \mathfrak{t}_2, ib); \rho_3\}}(\text{aWhile})$$

$$\frac{\begin{array}{c} \Gamma'' = \Gamma'[C_{imp} :: \mathfrak{e} \mathrel{+}= C_{imp} :: \mathfrak{E}_w] \quad \Delta \vdash \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \\ \Delta \vdash \langle S_1; \textbf{while}_{ib} \ e \ \textbf{do} \ S_1 \ \textbf{end while}, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle \quad n \neq 0 \end{array}}{\Delta \vdash \langle \textbf{while}_{ib} \ e \ \textbf{do} \ S_1 \ \textbf{end while}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle}(\text{eWhile-True})$$

From Theorem 5 and Theorem 4, knowing that reset function and worst case iteration preserve the ordering we get that $(\Gamma_2; \mathfrak{t}_1) \geq (\Gamma'; \mathfrak{t}')$ and moreover $\Gamma_2 \geq \Gamma'$.
Since we are not modifying any state and just add same amount of energy and timing, we derive that $(\Gamma_3; \mathfrak{t}_1 + C_{imp} :: \mathfrak{T}_w) \geq (\Gamma''; \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w)$ and $\Gamma_2 \geq \Gamma''$.
Just for the purpose of proof we create the new derivation $\{\Gamma''; \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma'_3; \mathfrak{t}'_3; \rho_2\}$ and we apply induction hypothesis with $\Delta \vdash \langle S_1; \textbf{while}_{ib}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma', \Gamma'', \mathfrak{t}' + C_{imp} :: \mathfrak{T}_w \rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}'' \rangle$.
We derive that $(\Gamma'_3; \mathfrak{t}'_3) \geq (\Gamma'''; \mathfrak{t}'')$. By theorem 5 we get also that $\Gamma'_3 \geq \Gamma'''$.
By lemma 4 we conclude that $(\Gamma_4; \mathfrak{t}_2) \geq (\Gamma'''; \mathfrak{t}'')$ and by lemma 2 we get also that $\Gamma_4 \geq \Gamma'''$.
Notice that the fixpoint we calculated at the beginning retrieve a component state that is bigger than every component state at end of every cycle in the while-loop (in the semantic tree).
Hence it follows that for every single cycle the analysis we have made till now is a good overestimation.
By Axiom 2, by the "constant power usage" property and by Theorem 4 we are sure that energy analysis plus the sum of all the $td(C,)$ for every component in $\textbf{wci}(\Gamma_4; e; S)$ can be taken as an overbound for the all the cycles. (Remember that $C_i^{\textbf{wci}(\Gamma; e; S)} :: s = C_i^{\Gamma_4} :: s$)
So, the final estimation for energy usage is define as the overbound for every cycle (what we have just calculated) times the number of cycles (the loop bound ib expresses that) plus the possible energy not take in account at the end of the loop.
Formally:

$$(C_i^{\textbf{wci}(\Gamma_4; e; S)} :: \mathfrak{e} - C_i^{\Gamma} :: \mathfrak{e}) + td(C_i^{\Gamma_4}, \mathfrak{t}_2)) * (ib) + C_i^{\Gamma} :: \mathfrak{e} =$$
$$(C_i^{\Gamma_4} :: \mathfrak{e} - C_i^{\Gamma} :: \mathfrak{e}) + td(C_i^{\Gamma_4}, \mathfrak{t}_2)) * (ib) + C_i^{\Gamma} :: \mathfrak{e}$$

The results seems similar to the energy computed but it is not. Indeed, the energy computed by the production rules lacks of $td(C_i^{\Gamma_4}, \mathfrak{t}_2) * (ib)$.
We have to revert the states at the beginning of the loop and by theorem 2 final timestamp is always bigger than the actual one. This assure us that the correspondent $td(C,)$ function retrieve us an energy consumption estimation that is like if the device would have remained in such state for all the while loop. Since power consumption is constant per state, the result is greater than $td(C_i^{\Gamma_4}, \mathfrak{t}_2) * (ib)$ (the sum of all the little delta not calculated in the energy). This is what the **oe** function does.

This concludes the proof. □

6 Conclusion and Future Work

We presented a hybrid, energy-aware Hoare logic for reasoning about energy consumption of systems controlled by software. The logic comes with an analysis which is proven to be sound with respect to the semantics. To our knowledge, our approach is the first attempt at bounding energy-consumption statically in a way which is parametric with respect to hardware models. This is a first step towards a hybrid approach to energy consumption analysis in which the software is analysed automatically together with the hardware it controls.

Future Work Many future research directions can be envisioned: e.g. providing an implementation of an automatic analysis for a real programming language[5], performing energy measurements for defining component models, modelling of software components and enabling the development of tools that can automatically derive energy consumption bounds for large systems, finding the most suited tool(s) to provide the right loop bounds and annotations for our analysis and study energy usage per time unit on systems that are always running, removing certain termination restrictions.

[5] A proof-of-concept implementation for our While language is available at http://resourceanalysis.cs.ru.nl/energy

References

1. Albers, S.: Energy-efficient algorithms. Commun. ACM **53**(5) (2010) 86–96
2. Saxe, E.: Power-efficient software. Commun. ACM **53**(2) (2010) 44–48
3. Ranganathan, P.: Recipe for efficiency: principles of power-aware computing. Commun. ACM **53**(4) (2010) 60–67
4. te Brinke, S., Malakuti, S., Bockisch, C., Bergmans, L., Akşit, M.: A design method for modular energy-aware software. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, New York, NY, USA, ACM (2013) 1180–1182
5. Cohen, M., Zhu, H.S., Senem, E.E., Liu, Y.D.: Energy types. SIGPLAN Not. **47**(10) (October 2012) 831–850
6. Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., Grossman, D.: Enerj: approximate data types for safe and general low-power computation. SIGPLAN Not. **46**(6) (June 2011) 164–174
7. Zhurikhin, D., Belevantsev, A., Avetisyan, A., Batuzov, K., Lee, S.: Evaluating power aware optimizations within GCC compiler. In: GROW-2009: International Workshop on GCC Research Opportunities. (2009)
8. Gheorghita, S.V., Corporaal, H., Basten, T.: Iterative compilation for energy reduction. J. Embedded Comput. **1**(4) (2005) 509–520
9. Kandemir, M., Vijaykrishnan, N., Irwin, M.J., Ye, W.: Influence of compiler optimizations on system power. In: Proceedings of the 37th Annual Design Automation Conference. DAC '00, New York, NY, USA, ACM (2000) 304–307
10. Junior, M.N.O., Neto, S., Maciel, P.R.M., Lima, R.M.F., Ribeiro, A., Barreto, R.S., Tavares, E., Braga, F.: Analyzing software performance and energy consumption of embedded systems by probabilistic modeling: An approach based on coloured petri nets. In: ICATPN'06. (2006) 261–281
11. Nogueira, B., Maciel, P., Tavares, E., Andrade, E., Massa, R., Callou, G., Ferraz, R.: A formal model for performance and energy evaluation of embedded systems. EURASIP J. Embedded Syst. (January 2011) 2:1–2:12
12. Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: COSTA: Design and Implementation of a Cost and Termination Analyzer for Java Bytecode. In: FMCO'07. Volume 5382 of LNCS., Springer (2008) 113–133
13. Hoffmann, J., Aehlig, K., Hofmann, M.: Multivariate amortized resource analysis. In: POPL'11, ACM (2011) 357–370
14. Atkey, R.: Amortised resource analysis with separation logic. In: ESOP. LNCS 6012 (2010) 85–103
15. Aspinall, D., Beringer, L., Hofmann, M., Loidl, H.W., Momigliano, A.: A program logic for resources. Theor. Comput. Sci. **389**(3) (December 2007) 411–445
16. Jayaseelan, R., Mitra, T., Li, X.: Estimating the worst-case energy consumption of embedded software. In: RTAS'06, IEEE (2006) 81–90
17. Kerrison, S., Liqat, U., Georgiou, K., Mena, A.S., Grech, N., Lopez-Garcia, P., Eder, K., Hermenegildo, M.V.: Energy consumption analysis of programs based on XMOS ISA-level models. In: LOPSTR'13, Springer (September 2013)
18. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D.B., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P.P., Staschulat, J., Stenström, P.: The worst-case execution-time problem - overview of methods and survey of tools. ACM Trans. Embedded Comput. Syst. **7**(3) (2008)
19. Shkaravska, O., van Eekelen, M.C.J.D., van Kesteren, R.: Polynomial size analysis of first-order shapely functions. Logical Methods in Comp. Sc.. **5**(2) (2009) 1–35
20. Shkaravska, O., Kersten, R., Van Eekelen, M.: Test-based inference of polynomial loop-bound functions. In: PPPJ'10: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, ACM (2010) 99–108
21. Kersten, R., van Gastel, B.E., Shkaravska, O., Montenegro, M., van Eekelen, M.: ResAna: a resource analysis toolset for (real-time) JAVA. Concurrency Computat.: Pract. Exper. (2013)
22. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Verification, Model Checking, and Abstract Interpretation. Volume 2937 of Lecture Notes in Computer Science. Springer (2004) 465–486
23. Hunt, J.J., Tonin, I., Siebert, F.: Using global data flow analysis on bytecode to aid worst case execution time analysis for real-time java programs. In Bollella, G., Locke, C.D., eds.: JTRES. Volume 343 of ACM International Conference Proceeding Series., ACM (2008) 97–105
24. Parisen Toldin, P., Kersten, R., van Gastel, B., van Eekelen, M.: Soundness Proof for a Hoare Logic for Energy Consumption Analysis. Technical Report ICIS–R13009, Radboud University Nijmegen (October 2013)
25. Anastasi, G., Conti, M., Francesco, M.D., Passarella, A.: Energy conservation in wireless sensor networks: A survey. Ad Hoc Networks **7**(3) (2009) 537 – 568