Soundness Proof for a Hoare Logic for Energy Consumption Analysis*

Technical Report Radboud University Nijmegen

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, School of Computer Science, The Netherlands
{bernard.vangastel,marko.vaneekelen}@ou.nl

Abstract. Hardware consumes energy. Software controls hardware. So, when the hardware is given, it is the software that determines how much energy is consumed. Energy inefficient software implementations cause battery drain for small systems and high energy costs for large systems. Energy analysis is often dynamic, via testing techniques. A static analysis that makes predictions about energy consumption can help in preventing such problems in a more cost-effective way.
Many kinds of analysis exist that are dedicated to a certain piece of hardware. In contrast, we propose a Hoare logic for energy consumption analysis that is parametric with respect to the hardware it controls. Energy models of hardware components can be specified separately from the logic. Parametrised with one or more of such component models, the logic can statically estimate energy-usage of the hardware controlled by the analysed software.
An energy-aware Hoare logic is presented. Soundness is proven and an example analysis is given for software controlling a wireless sensor node.

1 Introduction

Power consumption and green computing are nowadays among the most important topics in IT. From small systems such as a 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 on the costs of powering the device. These costs are often amplified by inefficient power-supplies and cooling of the system.

It is clear that power consumption depends not only on hardware, but also on the software *controlling* the hardware. Currently, the only method to analyse energy consumption of software available to programmers is dynamic analysis: running the software and measuring. Measuring power consumption of a system and especially of its individual components is not a trivial task. For designated measuring set-up is required. This means that most programmers have no idea whatsoever about the amount of energy that their software consumes. An automatic static analysis would be a big improvement, potentially leading to more energy-efficient software. Such a static analysis is presented in this paper. We are currently not aware of any other such method.

Since the software interacts with multiple components (software and hardware), energy consumption analysis needs to mix different kinds of analysis. Power consumption may depend on hardware state, values of variables and estimation of clock-cycles. Also, complexity plays a fundamental role in energy consumption estimation. An algorithm which runs in exponential time requires much more energy than an algorithm running in polynomial time.

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

Related Work There is a large body of work on energy-efficiency of software. Most papers that approach the problem on a higher level define programming and design patterns for writing energy-efficient code [1–3]. This is done specifically for wireless sensor networks in [4] and [5]. In [6], a modular design for energy-aware software is presented that is based on a series of rules on UML schemes. Similar works, dealing with the language itself and programming style, are the ones presented in [7–9], where the program is divided into "phases" describing similar behaviour. Hence, by knowing the software its behaviour, optimisations are proposed at design level for reaching lower energy consumption. A lot of research is dedicated to building compilers that optimize code for energy-efficiency [10–12]. In [13] and [14], energy modeling techniques are proposed based on Petri-nets. The processor is the only component that is modeled.

Our Approach The above approaches are all aimed at developing green software, mostly through energy-efficient programming patterns. We are interested in making actual estimations of the energy-consumption of software, which is more generally applicable. Our approach is new and it has not yet been investigated as such in the literature. Of course, it is strictly connected with resource analysis in general. Analysing power consumption is more difficult than timing analysis or space analysis since both hardware and software considerations have to be taken into account. Energy consumption models of hardware components are assumed to be input for our analysis. Of course, we need to have information about the software, such as dependencies between variables and information about the number of iterations a loop will make. For this reason we assume that a previous analysis (properly instantiated for our case) has been made retrieving ranking functions (for instance [15, 16]) and variable dependency information (for instance [17]).

Estimation of power usage requires to merge different kinds of analysis, since it may depend on many different things: time, space, program flow and power state of devices. We have earlier experience with research in size [18], loop bound [19, 16] and memory [16] analysis.

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. These can be both hardware and software component models. The central control is however assumed to be defined in software. Consequently, the analysis is done on a hybrid system of both software and hardware component models. The Hoare logic not only specifies which judgements can be used for reasoning about programs. It also has an approximation built in such that the reasoning can yield an upper bound for the energy consumption of a system of components that are controlled by software. In that sense it resembles a type system.

Our contribution The main contributions of this paper are:

- A new way of energy-aware modelling of systems with hardware and software components.
- An energy-aware semantics for the resulting hybrid models.
- An energy-aware Hoare logic that enables both formal reasoning about energy consumption and 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 logic 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. The example is given in Sect. 4 and the soundness proof is given in Sect. 5. The paper is concluded in Sect. 6.

2 Modelling Hybrid Systems

Information Technology systems consist of hardware and software components. In order to study the energy consumption of hybrid IT systems it is necessary to first study both kinds of components in one single modelling framework. This section defines a hybrid logic in which a software plays a central role controlling hardware components. The hardware components[4] are modelled in such a way that the relevant information for the energy consumption analysis is present. Other aspects of the hardware components are discarded. In this paper the controlling software is taken to be written in a small language designed just for illustrating the analysis.

[4] In fact, larger software components may be modelled in a similar way.

2.1 Language

Our analysis is performed on a 'while' language. As we use this language for illustration purposes, the only supported type in the language is an integer type. There are no global variables and parameters are passed by-value, so functions in our language do not have side-effects on the program state. As we already support `while` expressions we only support non-recursive functions. In our language we introduce 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 return an integer.

The grammar for our language is defined as follows:

$$
\begin{aligned}
c \in \textsc{Const} &= n \in \mathbb{N} \\
X \in \textsc{Var} &= X_1 \mid X_2 \mid X_3 \mid \ldots \\
e \in \textsc{Expr} &= c \mid X \mid X = e \mid e_1 \boxdot e_2 \mid C_i :: f(\mathit{args}) \mid f(\mathit{args}) \\
S \in \textsc{Statement} &= \textbf{skip} \mid S_1; S_2 \mid e \mid \textbf{if } e \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ end if} \\
&\quad \mid \textbf{while } e \textbf{ do } S \textbf{ end while} \\
\mathit{fundef} \in \textsc{Func} &= \textbf{function } \mathit{name}\,(\mathit{args})\ S \textbf{ return } X; \textbf{ end function}
\end{aligned}
$$

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

2.2 Modelling Components

In order to be able to correctly reason about hybrid systems we need some way to model hardware components that captures the behaviour (in terms of the effects on the program state) of those components. So we introduce a *component model* in this section. As we want to be able to reason about a broad range of components, we will use finite state models only. This enables us to be flexible in the devices we support. A component state $C_i :: s$ is a collection of variables. A variable v in the state of component C_i can be accessed with the expression $C_i :: s.v$. Variables in the model use the same type as program variables. They are all integers.

Another core aspect of our way of modelling is the inability of component models to influence each other. Because of this, all the state changes in components are explicit in the source code as an operation on that specific component. This aspect is also expressed in the definition of the update function for the component state. Such a function should be defined in the model for every component function. The update function is defined by $C_i :: \delta_f(s, \mathit{args}) = s'$: it takes the state and the arguments passed to the component function and returns the new state of the component.

2.3 Semantics

Standard, non-energy-aware semantics can be defined for our language. Below, only the assignment rule (sA) and the component function call rule (sCF) are given to illustrate the notation and the way components are dealt with. 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. As usual we also have substitution: let σ be a program state function, then $\sigma[n/X_i]$ represents a state function such that, on every $X_j \neq X_i$, $\sigma[n/X_i](X_j) = \sigma(X_j)$, otherwise $\sigma[n/X_i](X_i) = n$. 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.

$$
\frac{\langle e, \sigma, \Gamma \rangle \Downarrow^e \langle n, \sigma', \Gamma' \rangle}{\langle X_1 = e, \sigma, \Gamma \rangle \Downarrow^e \langle n, \sigma'[n/X_1], \Gamma'' \rangle} (\text{sA})
$$

Each component C_i may have multiple component functions. With $C_i :: f$ we denote selection of a particular function of a component. A component function is split into two parts, the part that produces the return value (rv_f) and the part that expresses the update of the internal state of the component (δ_f).

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

There are only integers: no explicit booleans. The value 0 is handled as a **False** value, while all the other possible values are handled as a **True** value. Notice that expressions are also statements. If they are used as statements we discard the result of the expression. The full semantics are given in Fig. 1.

$$\frac{}{\langle c, \sigma, \Gamma\rangle \Downarrow^e \langle c, \sigma, \Gamma\rangle}(\text{sN}) \quad \frac{}{\langle X, \sigma, \Gamma\rangle \Downarrow^e \langle \sigma(X), \sigma, \Gamma\rangle}(\text{sV})$$

$$\frac{\langle e_1, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle \quad \langle e_2, \sigma', \Gamma'\rangle \Downarrow^e \langle m, \sigma'', \Gamma''\rangle \quad e_1 \boxdot e_2 = p}{\langle e_1 \boxdot e_2, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle p, \sigma'', \Gamma''', \mathfrak{t}'' + C_{cpu} :: \mathfrak{T}_e\rangle}(\text{sE})$$

$$\frac{\langle e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle}{\langle X_1 = e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma[n/X_1], \Gamma''\rangle}(\text{sA})$$

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

$$\frac{\begin{array}{c}\langle S, \sigma^0[n_1, n_2, \ldots/X_1, X_2, \ldots], \Gamma\rangle \Downarrow^s \langle \sigma^1, \Gamma^1\rangle \quad \sigma^0 \text{ fresh} \\ \textbf{function } name\,(X_1, X_2 \ldots)\ S \textbf{ return } X_i\textbf{; end function} \in def\_functions\end{array}}{\langle name(n_1, n_2 \ldots), \sigma, \Gamma\rangle \Downarrow^e \langle \sigma'(X_i), \sigma, \Gamma'\rangle}(\text{sLF})$$

$$\frac{\langle e_1, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle}{\langle e_1, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle}(\text{sEasS}) \quad \frac{}{\langle \textbf{skip}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma, \Gamma\rangle}(\text{sS})$$

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

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

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

$$\frac{\langle e, \sigma, \Gamma\rangle \Downarrow^e \langle 0, \sigma', \Gamma'\rangle}{\langle \textbf{while}_{rf}\ e \textbf{ do } S_1 \textbf{ end while}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma', \Gamma'\rangle}(\text{sWF})$$

$$\frac{n \neq 0 \quad \langle e, \sigma, \Gamma\rangle \Downarrow^e \langle n, \sigma', \Gamma'\rangle \quad \langle S_1; \textbf{while}_{rf}\ e \textbf{ do } S_1 \textbf{ end while}, \sigma', \Gamma'\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}{\langle \textbf{while}_{rf}\ e \textbf{ do } S_1 \textbf{ end while}, \sigma, \Gamma\rangle \Downarrow^s \langle \sigma'', \Gamma''\rangle}(\text{sWT})$$

Fig. 1. Semantics for the `while` language.

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 representing one operation that will draw a constant amount of energy (disregarding any time aspect), the latter signifying turning a device on (or moving to a specific power state). While a device is in a certain power state the component will draw a constant amount of energy *per time unit*.

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 will be 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 in the component. For time-dependent energy usage, with each state change of the component, the energy used while the component was in the previous state is added to the accumulator. To this end, we need to track the time spent in the current state of the component. We add τ to the component, signifying the timestamp at which the component entered the current state. As mentioned before, we assume that each component has

a constant *power draw* while in a state. This is expressed in the component model by the function $C_i :: \phi(s)$, which maps component states onto the corresponding power draw. To calculate the power consumed while in a certain state we can define the global *td* function, with two arguments: the component and the current timestamp. The definition of *td* is:

$$td(c, t') = c :: \phi(s) * (t' - c :: t)$$

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

The new (sCF) rule for component functions is defined below, with $C_i :: \mathfrak{T}_f$ representing the time it costs to execute this component function. Hence the rule is modified in the following way:

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

As follows from this definition is that the energy accumulator of the components is not up to date with respect to the current time, as it is only updated in the sCF rule. This is done for simplicity; otherwise each rule that adjusts the global time needs to update the energy accumulator of all the components.

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

Important to note is our intention that *everything is a component*. This includes the processor, which can be used to capture the effect of executing a statement in our language, e.g. an arithmetical operation. The processor component C_{cpu} is, however, an integral part of our energy-aware semantics and logic. For simplicity, we assume the component C_{cpu} does not have a state. The CPU component should at least have resource consumption constants defined for the following operations (see Sect. 4 for an example of a minimal C_{cpu} definition): evaluation of an expression or integer comparison, assignment, evaluation of a while loop, evaluation of a conditional.

In order to capture the effect of these operations, we extend the rules signifying these operations. The extended (sA) rule for assignment is listed below, with $C_{cpu} :: \mathfrak{E}_a$ for the energy usage of an assignment and $C_{cpu} :: \mathfrak{T}_a$ for the time it takes to perform an assignment.

$$\frac{\langle e, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}' \rangle \quad \Gamma'' = \Gamma'[C_{cpu} :: \mathfrak{e} \mathrel{+}= C_{cpu} :: \mathfrak{E}_a]}{\langle X_1 = e, \sigma, \Gamma, \mathfrak{t} \rangle \Downarrow^e \langle n, \sigma'[n/X_1], \Gamma'', \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_a \rangle} (\text{sA})$$

We will 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 ranking function: $\mathbf{while}_{rf}$. We already added the *rf* to each while loop in our overview of the semantics, to make this assumption explicit. The energy-aware semantics can be found in Figure 2.

3.2 Energy Aware Modelling

In order for the analysis to work we need to assume that components are modelled in such a way that the component states reflect different power-levels and have an accompanying order. Greater states should imply greater power draw.

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. Because states are partially ordered, we can define a least-upper bound function.

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.

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 $C_i :: \phi(s_1) \geq C_i :: \phi(s_2)$.*

$$\frac{}{\langle c,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle c,\sigma,\Gamma,\mathfrak{t}\rangle}(\text{sN}) \qquad \frac{}{\langle X,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle \sigma(X),\sigma,\Gamma,\mathfrak{t}\rangle}(\text{sV})$$

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

$$\frac{\langle e,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle n,\sigma',\Gamma',\mathfrak{t}'\rangle \quad \Gamma''=\Gamma'[C_{cpu}::\mathfrak{e} \mathrel{+}= C_{cpu}::\mathfrak{E}_a]}{\langle X_1=e,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle n,\sigma'[n/X_1],\Gamma'',\mathfrak{t}'+C_{cpu}::\mathfrak{T}_a\rangle}(\text{sA})$$

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

$$\frac{\langle S,\sigma^0[n_1,n_2,\ldots/X_1,X_2,\ldots],\Gamma,\mathfrak{t}\rangle \Downarrow^s \langle\sigma^1,\Gamma^1,\mathfrak{t}^1\rangle \quad \sigma^0 \text{ fresh} \quad \textbf{function } name\,(X_1,X_2\ldots)\ S \textbf{ return } X_i\textbf{; end function} \in def\_functions}{\langle name(n_1,n_2\ldots),\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle\sigma'(X_i),\sigma,\Gamma',\mathfrak{t}'\rangle}(\text{sLF})$$

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

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

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

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

$$\frac{\langle e,\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^e \langle 0,\sigma',\Gamma',\mathfrak{t}'\rangle \quad \Gamma''=\Gamma'[C_{cpu}::\mathfrak{e} \mathrel{+}= C_{cpu}::\mathfrak{E}_w]}{\langle \textbf{while}_{rf}\ e \textbf{ do } S_1 \textbf{ end while},\sigma,\Gamma,\mathfrak{t}\rangle \Downarrow^s \langle\sigma',\Gamma'',\mathfrak{t}'+C_{cpu}::\mathfrak{T}_w\rangle}(\text{sWF})$$

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

Fig. 2. Energy-aware semantics.

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

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

3.3 A Hoare Logic for Energy Analysis

This section covers two things at once. Firstly, it treats the definition of an energy-aware logic that follows the energy aware semantics as much as possible and secondly it gives energy analysis rules that can be used to show that the total energy consumption of the analysed system will be below a certain upper bound. Both are defined at once in one single set of production rules. This full set of production rules is given in Fig. 3.

Making an energy consumption analysis of a program is surely a challenging task; the problem is strictly connected with many other problems such as loop analysis and symbolic analysis of variables. First, one needs to know how variables are correlated to each other. This is not the subject of this paper. For that reason we assume an earlier applied analysis based on Hoare Logic, which gives us a symbolic state of every variable for each line of code, e.g, $\{X_1 = e_1\}X_1 = X_1 + X_2 + X_4\{X_1 = e_1 + X_2 + X_4\}$, plus other non-energy related properties (invariants, ranking functions) that have already been proven. These properties are collected in a symbolic state function ρ.

All the judgements in the production rules have the following shape: $\{\Gamma; \mathfrak{t}; \rho\}S\ \{\Gamma'; \mathfrak{t}'; \rho'\}$, where ρ represents the symbolic state function retrieved from the earlier standard analysis. We use the notation $\Gamma[n \leftarrow n_{new}]$ to mean a copy of Γ with n replaced by n_{new}. Also we use $\Gamma[n \mathrel{+}= m]$ to mean $\Gamma[n \leftarrow n + m]$. As said before, we require to have an upper-bound on the number of cycles of the while-loop, expressed by a ranking function *rf* labelling the `while` statement. As states are partially ordered, we can take a least upper bound of states $\mathbf{lub}(s_1, s_2)$ or sets of component states $\mathbf{lub}(\Gamma_1, \Gamma_2)$.

We want to point the user to the aspects of the production rules that are most relevant for the analysis. The rule (CFtd) uses the $td(C_i, t)$ function to estimate the time dependent energy consumption of the calls of component functions as explained above. The if-then-else rule (I) takes the least upper bound of the energy aware component states and the maximum of the time estimates. The CPU-rules (E) and (A) keep track of the energy consumption of the CPU and the if-then-else and while rules also adjust the CPU its energy consumption.

Special attention is warranted for the while rule (W). It overestimates the time-dependent energy consumption by estimating the highest energy-aware component state that can be achieved during the loop and assuming that the component has been in that state throughout the loop. Also, it multiplies the resource consumption in a *worst-case* iteration by the (possibly overestimated) number of iterations. Three calculations are needed:

1. Energy usage function $\mathbf{e}$. After the loop we have to overestimate energy consumption by taking the difference between the accumulated usage before and after one iteration and multiplying this with the number of iterations (i.e. the ranking function). The function $\mathbf{e}(\Gamma_{out}, \Gamma_{in}, rf)$ calculates for every component $C_i^{\Gamma_{in}} :: \mathfrak{e} + (C_i^{\Gamma_{out}} :: \mathfrak{e} - C_i^{\Gamma_{in}} :: \mathfrak{e}) * rf$.
2. Regress Function $\mathbf{r}$. To overestimate time-dependent energy usage, we backdate 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 in this state since right before entering the loop. The regress function $\mathbf{r}(C_i, \mathfrak{t})$ sets the value of $C_i :: \tau$ to $\mathfrak{t}$ if $C_i :: \tau > \mathfrak{t}$.
3. Fixpoint Function $\mathbf{fix}$. 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. We introduce the fixpoint function $\mathbf{fix}_i(C_i :: s, S)$ to find this maximum input state. Let $f_i : s \to s$ be a function representing the concatenation of all the $C_i :: \delta_f()$ applied in S. By f_i^n we mean $f_i \circ f_i^{n-1}$, with $f_i^1 = f_i$. Now find n such that there exists $k < n$ for which $f_i^n(s) = f_i^k(s)$. The fixpoint function can now be defined as $\mathbf{fix}_i(C_i :: s, S) = \mathbf{lub}(s, f_i(s), f_i^2(s), \ldots, f_i^n(s))$. The global fixpoint function $\mathbf{fix}(\Gamma, S)$ is defined by iteratively applying the fixpoint function of each component.

Applying the production rules gives an estimate of the sum of the incidental energy consumption and the time dependent energy consumption. However, the time-dependent energy consumption is only counted at changes of component states. So, 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 thus be calculated by $\mathfrak{e}_{system}(\Gamma_{end}, \mathfrak{t}_{end})$ where $\mathfrak{e}_{system}$ is defined as follows:

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

$$\frac{}{\{\Gamma;\mathfrak{t};\rho\}n\{\Gamma;\mathfrak{t};\rho\}}(\mathrm{N}) \qquad \frac{}{\{\Gamma;\mathfrak{t};\rho\}X\{\Gamma;\mathfrak{t};\rho\}}(\mathrm{V})$$

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

$$\frac{\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \quad \Gamma_2 = \Gamma_1[C_{cpu} :: \mathfrak{e} \mathrel{+}= C_{cpu} :: \mathfrak{E}_a]}{\{\Gamma;\mathfrak{t};\rho\}X = e\{\Gamma_2;\mathfrak{t}_1 + C_{cpu} :: \mathfrak{T}_a;\rho_2\}}(\mathrm{A})$$

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

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

$$\frac{\begin{array}{l}\{\Gamma;\mathfrak{t};\rho\}S\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \\ \textbf{function } name\,(X_1,\ldots,X_n)\ S \textbf{ return } X\textbf{; end function} \in def\_functions\end{array}}{\{\Gamma;\mathfrak{t};\rho\}name(X_j,\ldots,X_{j+n})\{(\Gamma_1;\mathfrak{t}_1)[X_1,\ldots,X_n/\rho(X_j,\ldots,X_{j+n})];\rho\}}(\mathrm{LF})$$

$$\frac{}{\{\Gamma;\mathfrak{t};\rho\}\textbf{skip}\{\Gamma;\mathfrak{t};\rho\}}(\mathrm{S}) \qquad \frac{\{\Gamma;\mathfrak{t};\rho\}S_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \quad \{\Gamma_1;\mathfrak{t}_1;\rho_1\}S_2\{\Gamma_2;\mathfrak{t}_2;\rho_2\}}{\{\Gamma;\mathfrak{t};\rho\}S_1;S_2\{\Gamma_2;\mathfrak{t}_2;\rho_2\}}(\mathrm{C})$$

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

$$\frac{\begin{array}{cc}\multicolumn{2}{c}{\Gamma_2 = \Gamma_1[C_{cpu} :: \mathfrak{e} \mathrel{+}= C_{cpu} :: \mathfrak{E}_w]} \\ \{\textbf{fix}(\Gamma,e;S);\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} & \{\Gamma_2;\mathfrak{t}_1 + C_{cpu} :: \mathfrak{T}_w;\rho_1\}S\{\Gamma_3;\mathfrak{t}_2;\rho_2\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\textbf{while}_{rf}\ e \textbf{ do } S \textbf{ end while}\{\mathbf{e}(\mathbf{r}(\Gamma_3,\mathfrak{t}),\Gamma,rf),\mathfrak{t} + ((\mathfrak{t}_2 - \mathfrak{t}) * rf);\rho_3\}}(\mathrm{W})$$

Fig. 3. Production rules for statements.

3.4 Building Hardware Models

In this section we list some instructions for building hardware models.

CPU component It is assumed that at least one component is always part of Γ, namely the CPU component C_{cpu}. This component model is used inside the type system of our analysis to calculate the time and energy consumption of arithmetic operations, assignments and control structures. It should therefore have (at least) the following constants defined:

- $C_{cpu} :: \mathfrak{E}_e$ and $C_{cpu} :: \mathfrak{T}_e$ for evaluation of arithmetic expressions and integer comparison.
- $C_{cpu} :: \mathfrak{E}_a$ and $C_{cpu} :: \mathfrak{T}_a$ for assignment.
- $C_{cpu} :: \mathfrak{E}_w$ and $C_{cpu} :: \mathfrak{T}_w$ for while loops.
- $C_{cpu} :: \mathfrak{E}_{ite}$ and $C_{cpu} :: \mathfrak{T}_{ite}$ for conditionals.

For simplicity, we assume a stateless CPU that does not consume energy based on time. In other words, its time-based energy usage is incorporated in the incidental energy usage given by the resource consumption functions above. It would not be a complex extention to add time-based consumption. Every production rule where one of the above constants is used should be extended with a call to the *td* function and an update of $C_{cpu} :: \tau$, as in the CF rule. This is omitted in this paper for simplicity of presentation.

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

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 (W) rule, we cannot simply analyse one iteration, but need to find the fixpoint first.

	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.

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[5].

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 (I) 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.

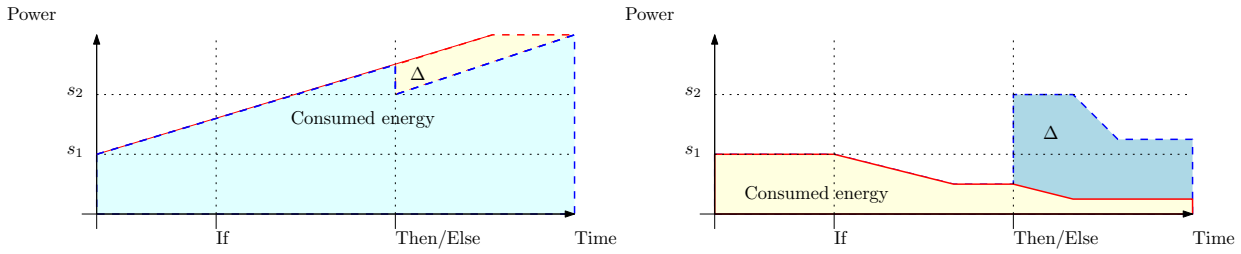

Fig. 4. Increasing and decreasing power consumption in a conditional.

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:

[5] 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

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 (W) 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.

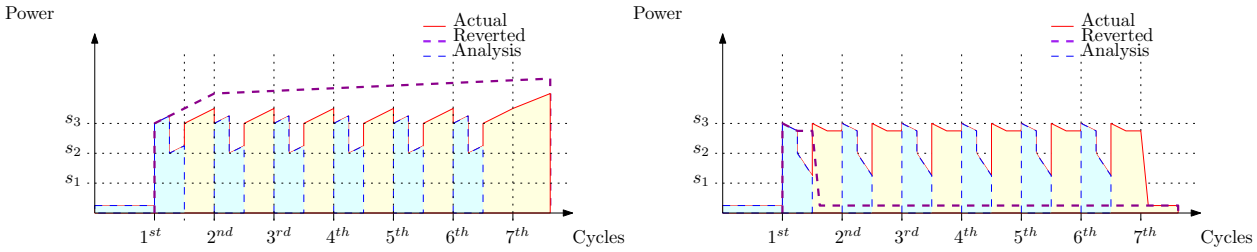

Fig. 5. Increasing and decreasing power consumption in a while.

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 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 (W) 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

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.

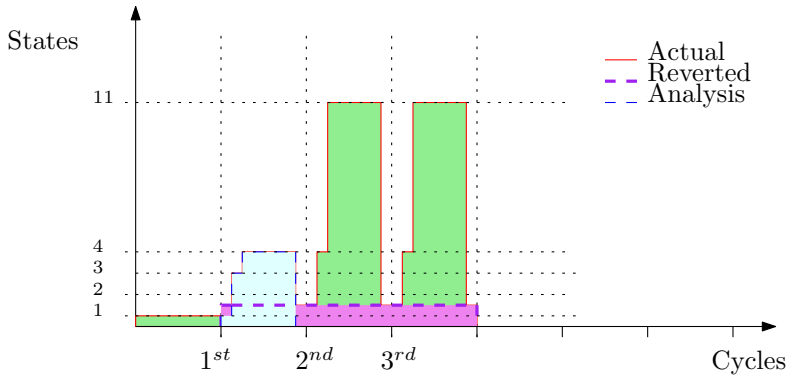

Fig. 6. Possible execution of the example program.

4 Example: Wireless Sensor Node

As an example for our analysis, we have modeled a wireless sensor node (WSN) with three components: a processor, a 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 three components: the CPU, a temperature sensor and the radio.

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

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

Table 2. The C_{cpu} 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 amortizing 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	$C_r :: \mathfrak{T}_{on} = 40,\ C_r :: \mathfrak{T}_{off} = 20,\ C_r :: \mathfrak{T}_{queue} = 30,\ C_r :: \mathfrak{T}_{send} = 100,$ $C_r :: \mathfrak{E}_{on} = 40,\ C_r :: \mathfrak{E}_{off} = 20,\ C_r :: \mathfrak{E}_{queue} = 30,\ C_r :: \mathfrak{E}_{send} = 100$
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. Algorithm 1 sends the results of all measurements out individually. Algorithm 2 collects the results of 10 measurements and sends them in a single message (if n is not a multiple of 10, it takes a bit more measurements than required). The latter is an example of *duty-cycling*, which is a well-known method for energy conservation in wireless sensor networks [5]. Our analysis indeed gives the expected result: Algorithm 2 is more energy-efficient.

```
function ALGORITHM1(X_n)
    C_r.on();
    while_{X_n} X_n > 0 do
        X_1 = C_s.m();
        C_r.queue(X_1);
        C_r.send();
        X_n = X_n − 1
    end while;
    C_r.off();
return 0;
end function
```

```
function ALGORITHM2(X_n)
    while_{⌈X_n/10⌉} X_n > 0 do
        X_i = 10;
        while_{X_i} X_i > 0 do
            X_1 = C_s.m();
            C_r.queue(X_1);
            X_i = X_i − 1;
            X_n = X_n − 1
        end while;
        C_r.on();
        C_r.send();
        C_r.off()
    end while
return 0;
end function
```

Algorithm 1 The evaluation of the states in the application of the logic to Algorithm 1 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 (CF) 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 C rule again, then continue with the analysis of the while loop. First, we must analyse the guard. The variable (V) and constant (N) rules do not affect energy or time consumption. The rule for evaluating an expression (E) adds the constant time and energy costs for the CPU. Then, the constant incidental resource consumption is added for the while loop.

We then analyse the loop body, which starts with a sensor measurement. The method only incidentally consumes resources, so only constants have to be added (there is no state to change). Then, the value is assigned, so the corresponding constants are added to the global time and the energy consumption of the CPU.

Then, the measurement is queued. This does not change the state of the radio, so only constants have to be added to the consumptions. Sending the queue also takes constant time and energy in our model. 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 of the CPU.

We now have to overestimate, from the evaluation of the loop body, the state after the whole loop. Because all the component states before the analysis of the loop-body are equal to those after the analysis of the body, the **fix**() functions reduce to the identity function here. We can thus, for each resource consumption variable

Fig. 7. Analysis for algorithm 1, $\mathbf{P}_*$ represent partial proofs, i.e. they refer to other parts of the figure.

calculate the difference in value before and after the loop body, then multiply this by the ranking function of the loop $\rho_0(X_n)$ (we need to use the value of X_n at the point right before the loop). For the result, see Table 5.

Finally, we turn the radio off. This means that the constant costs of $C_r::on()$ must be added. Moreover, as the state of the radio is changed, its time-dependent energy costs must be calculated. These are $(\mathfrak{t}_\mathbf{0} + 40 + 195 \cdot \rho_0(X_n) - \mathfrak{t}_\mathbf{0}) \cdot (2 + 200 \cdot 1)$, which can be simplified to $8080 + 39390 \cdot \rho_0(X_n)$.

After applying the Hoare logic, we still need to add the time-dependent energy-consumption for each component. The CPU its time-dependent consumption is already amortized in its incidental energy consumptions (although it would be simple to add time-dependent consumption if needed). Its energy consumption at Γ_{11} is thus final. The result for the sensor is $\mathfrak{e}^\mathbf{s}_\mathbf{0} + 40 \cdot \rho_0(X_n) + ((\mathfrak{t}_\mathbf{0} + 60 + 200 \cdot \rho_0(X_n)) - \mathfrak{t}_\mathbf{0}) \cdot 3$, which simplifies to $\mathfrak{e}^\mathbf{s}_\mathbf{0} + 120 + 640 \cdot \rho_0(X_n)$. The result for the radio is $\mathfrak{e}^\mathbf{r}_\mathbf{0} + 8120 + (\mathfrak{t}_\mathbf{0} - \tau^\mathbf{r}_\mathbf{0}) \cdot (2 + 200 \cdot \mathbf{on}^\mathbf{r}_\mathbf{0}) + 39520 \cdot \rho_0(X_n) + ((\mathfrak{t}_\mathbf{0} + 60 + 200 \cdot \rho_0(X_n)) - (\mathfrak{t}_\mathbf{0} + 40 + 200 \cdot \rho_0(X_n))) \cdot (2 + 200 \cdot 0)$, which simplifies to $\mathfrak{e}^\mathbf{r}_\mathbf{0} + 8160 + (\mathfrak{t}_\mathbf{0} - \tau^\mathbf{r}_\mathbf{0}) \cdot (2 + 200 \cdot \mathbf{on}^\mathbf{r}_\mathbf{0}) + 39520 \cdot \rho_0(X_n)$.

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

$$\mathfrak{e}^\mathbf{cpu}_\mathbf{0} + 60 \cdot \rho_0(X_n) + \mathfrak{e}^\mathbf{s}_\mathbf{0} + 120 + 640 \cdot \rho_0(X_n) + \mathfrak{e}^\mathbf{r}_\mathbf{0} + 8160 + (\mathfrak{t}_\mathbf{0} - \tau^\mathbf{r}_\mathbf{0}) \cdot (2 + 200 \cdot \mathbf{on}^\mathbf{r}_\mathbf{0}) + 39520 \cdot \rho_0(X_n) =$$

$$\mathfrak{e}^\mathbf{cpu}_\mathbf{0} + \mathfrak{e}^\mathbf{s}_\mathbf{0} + \mathfrak{e}^\mathbf{r}_\mathbf{0} + 8280 + 40090 \cdot \rho_0(X_n) + (\mathfrak{t}_\mathbf{0} - \tau^\mathbf{r}_\mathbf{0}) \cdot (2 + 200 \cdot \mathbf{on}^\mathbf{r}_\mathbf{0})$$

State	$\mathfrak{t}$	$C_{cpu} :: \mathfrak{e}$	$C_s :: \mathfrak{e}$	$C_r :: s.on$	$C_r :: \tau$	$C_r :: \mathfrak{e}$
$\Gamma_s, \mathfrak{t}_0$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_1, \mathfrak{t}_1$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 40 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_2, \mathfrak{t}_2$	$\mathfrak{t}_\mathbf{0} + 50$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 10$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 40 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_3, \mathfrak{t}_3$	$\mathfrak{t}_\mathbf{0} + 75$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 35$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 40 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_4, \mathfrak{t}_4$	$\mathfrak{t}_\mathbf{0} + 85$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 35$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 40 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_5, \mathfrak{t}_5$	$\mathfrak{t}_\mathbf{0} + 90$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 40 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_6, \mathfrak{t}_6$	$\mathfrak{t}_\mathbf{0} + 120$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 70 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_7, \mathfrak{t}_7$	$\mathfrak{t}_\mathbf{0} + 220$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 170 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_8, \mathfrak{t}_8$	$\mathfrak{t}_\mathbf{0} + 230$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 50$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 170 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_9, \mathfrak{t}_9$	$\mathfrak{t}_\mathbf{0} + 235$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 55$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 170 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_{10}, \mathfrak{t}_{10}$	$\mathfrak{t}_\mathbf{0} + 40 + 195 \cdot \rho_0(X_n)$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 55 \cdot \rho_0(X_n)$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_0(X_n)$	1	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 40 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 130 \cdot \rho_0(X_n)$
$\Gamma_{11}, \mathfrak{t}_{11}$	$\mathfrak{t}_\mathbf{0} + 60 + 195 \cdot \rho_0(X_n)$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 55 \cdot \rho_0(X_n)$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_0(X_n)$	0	$\mathfrak{t}_\mathbf{0} + 40 + 195 \cdot \rho_0(X_n)$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 8120 + (\mathfrak{t}_\mathbf{0} - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r}) + 39520 \cdot \rho_0(X_n)$

Table 5. Global states for Algorithm 1.

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:

$$8280 + 40090 \cdot \rho_0(X_n)$$

Algorithm 2 The evaluation of the states in the application of the logic to Algorithm 2 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 while rule. This adds the constant costs for evaluating an expression and for the while loop.

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 adds 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 12, multiplied by the ranking function $\rho_1(X_i)$.

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 consumption is added to its energy usage. The time-dependent energy usage in this case is $((\mathfrak{t}_\mathbf{0} + 215 + 75 \cdot \rho_1(X_i)) - (\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i))) \cdot (2 + 200 \cdot 1)$, which is equal to 28280.

We now need to overestimate the result of the outer loop. 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 (W) rule first calculates the maximal state in which the loop may actually be entered (the fixpoint). For $C_r :: s.on$ this gives the result $\max(0, \mathbf{on}_\mathbf{0}^\mathbf{r}) = \mathbf{on}_\mathbf{0}^\mathbf{r}$. It then multiplies this with the ranking function $\lceil \rho_1(X_n)/10 \rceil$. This gives an

$\mathbf{P_{outerguard}}$:

$$\frac{\overline{\{\Gamma_s;\mathfrak{t}_0;\rho_0\}X_n\{\Gamma_s;\mathfrak{t}_0;\rho_0\}}(\mathrm{V})\quad\overline{\{\Gamma_s;\mathfrak{t}_0;\rho_0\}0\{\Gamma_s;\mathfrak{t}_0;\rho_0\}}(\mathrm{N})\quad\Gamma_1=\Gamma_0[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_e]}{\{\Gamma_s;\mathfrak{t}_0;\rho_0\}X_n>0\{\Gamma_1;\mathfrak{t}_1=\mathfrak{t}_0+C_{cpu}::\mathfrak{T}_e;\rho_0\}}(\mathrm{E})$$

$\mathbf{P_i}$:

$$\frac{\overline{\{\Gamma_2;\mathfrak{t}_2;\rho_0\}10\{\Gamma_2;\mathfrak{t}_2;\rho_0\}}(\mathrm{N})\quad\Gamma_3=\Gamma_2[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_a]}{\{\Gamma_2;\mathfrak{t}_2;\rho_0\}X_i=10\{\Gamma_3;\mathfrak{t}_3=\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_a;\rho_1\}}(\mathrm{A})$$

$\mathbf{P_{innerguard}}$:

$$\frac{\overline{\{\Gamma_3;\mathfrak{t}_3;\rho_1\}X_i\{\Gamma_3;\mathfrak{t}_3;\rho_1\}}(\mathrm{V})\quad\overline{\{\Gamma_3;\mathfrak{t}_3;\rho_1\}0\{\Gamma_3;\mathfrak{t}_3;\rho_1\}}(\mathrm{N})\quad\Gamma_4=\Gamma_3[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_e]}{\{\Gamma_3;\mathfrak{t}_3;\rho_1\}X_i>0\{\Gamma_4;\mathfrak{t}_4=\mathfrak{t}_3+C_{cpu}::\mathfrak{T}_e;\rho_1\}}(\mathrm{E})$$

$\mathbf{P_m}$:

$$\frac{\Gamma_6=\Gamma_5[C_s::s\leftarrow C_s::\delta_m(C_s::s),C_s::\tau\leftarrow\mathfrak{t}_5,C_s::\mathfrak{e}\mathrel{+}=C_s::\mathfrak{E}_m+td(C_s,()\mathfrak{t}_5)]}{\{\Gamma_5;\mathfrak{t}_5;\rho_1\}C_s.m()\{\Gamma_6;\mathfrak{t}_6=\mathfrak{t}_5+C_s::\mathfrak{T}_m;\rho_1\}}(\mathrm{CF})\quad\frac{\Gamma_7=\Gamma_6[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_a]}{\{\Gamma_5;\mathfrak{t}_5;\rho_1\}X_1=C_s.m()\{\Gamma_7;\mathfrak{t}_7=\mathfrak{t}_6+C_{cpu}::\mathfrak{T}_a;\rho_2\}}(\mathrm{A})$$

$\mathbf{P_{queue}}$:

$$\frac{\Gamma_8=\Gamma_7[C_r::s\leftarrow C_r::\delta_{queue}(C_r::s),C_r::\tau\leftarrow\mathfrak{t}_7,C_r::\mathfrak{e}\mathrel{+}=C_r::\mathfrak{E}_{queue}+td(C_r,()\mathfrak{t}_7)]}{\{\Gamma_7;\mathfrak{t}_7;\rho_2\}C_r.queue()\{\Gamma_8;\mathfrak{t}_8=\mathfrak{t}_7+C_r::\mathfrak{T}_{queue};\rho_2\}}(\mathrm{CF})$$

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

$$\frac{\overline{\{\Gamma_8;\mathfrak{t}_8;\rho_2\}X_i\{\Gamma_8;\mathfrak{t}_8;\rho_2\}}(\mathrm{V})\quad\overline{\{\Gamma_8;\mathfrak{t}_8;\rho_2\}1\{\Gamma_8;\mathfrak{t}_8;\rho_2\}}(\mathrm{N})\quad\Gamma_9=\Gamma_8[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_e]}{\{\Gamma_8;\mathfrak{t}_8;\rho_2\}X_i-1\{\Gamma_9;\mathfrak{t}_9=\mathfrak{t}_8+C_{cpu}::\mathfrak{T}_e;\rho_2\}}\mathrm{E}$$

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

$$\frac{\mathbf{P_{i-1}}\quad\Gamma_{10}=\Gamma_9[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_a]}{\{\Gamma_8;\mathfrak{t}_8;\rho_2\}X_i=X_i-1\{\Gamma_{10};\mathfrak{t}_{10}=\mathfrak{t}_9+C_{cpu}::\mathfrak{T}_a;\rho_3\}}(\mathrm{A})$$

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

$$\frac{\overline{\{\Gamma_{10};\mathfrak{t}_{10};\rho_3\}X_n\{\Gamma_{10};\mathfrak{t}_{10};\rho_3\}}(\mathrm{V})\quad\overline{\{\Gamma_{10};\mathfrak{t}_{10};\rho_3\}1\{\Gamma_{10};\mathfrak{t}_{10};\rho_3\}}(\mathrm{N})\quad\Gamma_{11}=\Gamma_{10}[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_e]}{\{\Gamma_{10};\mathfrak{t}_{10};\rho_3\}X_n-1\{\Gamma_{11};\mathfrak{t}_{11}=\mathfrak{t}_{10}+C_{cpu}::\mathfrak{T}_e;\rho_3\}}\mathrm{E}$$

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

$$\frac{\mathbf{P_{n-1}}\quad\Gamma_{12}=\Gamma_{11}[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_a]}{\{\Gamma_{10};\mathfrak{t}_{10};\rho_3\}X_n=X_n-1\{\Gamma_{12};\mathfrak{t}_{12}=\mathfrak{t}_{11}+C_{cpu}::\mathfrak{T}_a;\rho_4\}}(\mathrm{A})$$

$\mathbf{P_{innerbody}}$:

$$\frac{\mathbf{P_m}\quad\dfrac{\mathbf{P_{queue}}\quad\dfrac{\mathbf{P_{i--}}\quad\mathbf{P_{n--}}}{\{\Gamma_8;\mathfrak{t}_8;\rho_2\}X_i=X_i-1;X_n=X_n-1\{\Gamma_{12};\mathfrak{t}_{12};\rho_4\}}(\mathrm{C})}{\{\Gamma_7;\mathfrak{t}_7;\rho_2\}C_r.queue(X_1);X_i=X_i-1;X_n=X_n-1\{\Gamma_{12};\mathfrak{t}_{12};\rho_4\}}(\mathrm{C})}{\{\Gamma_5;\mathfrak{t}_5=\mathfrak{t}_4+C_{cpu}::\mathfrak{T}_w;\rho_1\}X_1=C_s.m();C_r.queue(X_1);X_i=X_i-1;X_n=X_n-1\{\Gamma_{12};\mathfrak{t}_{12};\rho_4\}}(\mathrm{C})$$

$\mathbf{P_{inner}}$:

$$\frac{\mathbf{P_{innerguard}}\quad\Gamma_5=\Gamma_4[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_w]\quad\mathbf{P_{innerbody}}}{\{\Gamma_3;\mathfrak{t}_3;\rho_1\}\mathbf{while}_{X_i}\ X_i>0\ \mathbf{do}\ \ldots\ \mathbf{end\ while}\{\Gamma_{13};\mathfrak{t}_{13};\rho_5\}}(\mathrm{W})$$

$\mathbf{P_{send}}$:

$$\frac{\Gamma_{15}=\Gamma_{14}[C_r::s\leftarrow C_r::\delta_{send}(C_r::s),C_r::\tau\leftarrow\mathfrak{t}_{14},C_r::\mathfrak{e}\mathrel{+}=C_r::\mathfrak{E}_{send}+td(C_r,()\mathfrak{t}_{14})]}{\{\Gamma_{14};\mathfrak{t}_{14};\rho_5\}C_r.send()\{\Gamma_{15};\mathfrak{t}_{15}=\mathfrak{t}_{14}+C_r::\mathfrak{T}_{send};\rho_5\}}(\mathrm{CF})$$

$\mathbf{P_{radio2}}$:

$$\frac{\mathbf{P_{send}}\quad\dfrac{\Gamma_{16}=\Gamma_{15}[C_r::s\leftarrow C_r::\delta_{off}(C_r::s),C_r::\tau\leftarrow\mathfrak{t}_{15},C_r::\mathfrak{e}\mathrel{+}=C_r::\mathfrak{E}_{off}+td(C_r,()\mathfrak{t}_{15})]}{\{\Gamma_{15};\mathfrak{t}_{15};\rho_5\}C_r.off()\{\Gamma_{16};\mathfrak{t}_{16}=\mathfrak{t}_{15}+C_r::\mathfrak{T}_{off};\rho_5\}}(\mathrm{CF})}{\{\Gamma_{14};\mathfrak{t}_{14};\rho_5\}C_r.send();C_r.off()\{\Gamma_{16};\mathfrak{t}_{16};\rho_5\}}(\mathrm{C})$$

$\mathbf{P_{radio}}$:

$$\frac{\Gamma_{14}=\Gamma_{13}[C_r::s\leftarrow C_r::\delta_{on}(C_r::s),C_r::\tau\leftarrow\mathfrak{t}_{13},C_r::\mathfrak{e}\mathrel{+}=C_r::\mathfrak{E}_{on}+td(C_r,()\mathfrak{t}_{13})]}{\{\Gamma_{13};\mathfrak{t}_{13};\rho_5\}C_r.on()\{\Gamma_{14};\mathfrak{t}_{14}=\mathfrak{t}_{13}+C_r::\mathfrak{T}_{on};\rho_5\}}(\mathrm{CF})\quad\frac{\mathbf{P_{radio2}}}{\{\Gamma_{13};\mathfrak{t}_{13};\rho_5\}C_r.on();C_r.send();C_r.off()\{\Gamma_{16};\mathfrak{t}_{16};\rho_5\}}(\mathrm{C})$$

$\mathbf{P_{outerbody}}$:

$$\frac{\mathbf{P_i}\quad\dfrac{\mathbf{P_{inner}}\quad\mathbf{P_{radio}}}{\{\Gamma_3;\mathfrak{t}_3;\rho_1\}\mathbf{while}_{X_i}\ X_i>0\ \mathbf{do}\ \ldots\ \mathbf{end\ while};\ldots\{\Gamma_{16};\mathfrak{t}_{16};\rho_5\}}(\mathrm{C})}{\{\Gamma_2;\mathfrak{t}_2=\mathfrak{t}_1+C_{cpu}::\mathfrak{T}_w;\rho_0\}X_i=10;\mathbf{while}_{X_i}\ X_i>0\ \mathbf{do}\ \ldots\ \mathbf{end\ while};\ldots\{\Gamma_{16};\mathfrak{t}_{16};\rho_5\}}(\mathrm{C})$$

START :

$$\frac{\mathbf{P_{outerguard}}\quad\Gamma_2=\Gamma_1[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_w]\quad\mathbf{P_{outerbody}}}{\{\Gamma_s;\mathfrak{t}_0;\rho_0\}\mathbf{while}_{X_n}\ X_n>0\ \mathbf{do}\ \ldots\ \mathbf{end\ while}\{\Gamma_{17};\mathfrak{t}_{17};\rho_6\}}(\mathrm{W})$$

Fig. 8. Analysis for algorithm 2, $\mathbf{P}_*$ represent partial proofs, i.e. they refer to other parts of the figure.

State	$\mathfrak{t}$	$C_{cpu} :: \mathfrak{e}$	$C_s :: \mathfrak{e}$	$C_r :: s.on$	$C_r :: \tau$	$C_r :: \mathfrak{e}$
$\Gamma_s, \mathfrak{t}_0$	$\mathfrak{t}_\mathbf{0}$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu}$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_1, \mathfrak{t}_1$	$\mathfrak{t}_\mathbf{0} + 10$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 10$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_2, \mathfrak{t}_2$	$\mathfrak{t}_\mathbf{0} + 35$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 35$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_3, \mathfrak{t}_3$	$\mathfrak{t}_\mathbf{0} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 40$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_4, \mathfrak{t}_4$	$\mathfrak{t}_\mathbf{0} + 50$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 50$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_5, \mathfrak{t}_5$	$\mathfrak{t}_\mathbf{0} + 75$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75$	$\mathfrak{e}_\mathbf{0}^\mathbf{s}$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_6, \mathfrak{t}_6$	$\mathfrak{t}_\mathbf{0} + 85$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_7, \mathfrak{t}_7$	$\mathfrak{t}_\mathbf{0} + 90$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 80$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r}$
$\Gamma_8, \mathfrak{t}_8$	$\mathfrak{t}_\mathbf{0} + 120$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 80$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 30$
$\Gamma_9, \mathfrak{t}_9$	$\mathfrak{t}_\mathbf{0} + 130$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 90$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 30$
$\Gamma_{10}, \mathfrak{t}_{10}$	$\mathfrak{t}_\mathbf{0} + 135$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 95$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 30$
$\Gamma_{11}, \mathfrak{t}_{11}$	$\mathfrak{t}_\mathbf{0} + 145$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 105$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 30$
$\Gamma_{12}, \mathfrak{t}_{12}$	$\mathfrak{t}_\mathbf{0} + 150$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 110$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 30$
$\Gamma_{13}, \mathfrak{t}_{13}$	$\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75 + 35 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_1(X_i)$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 30 \cdot \rho_1(X_i)$
$\Gamma_{14}, \mathfrak{t}_{14}$	$\mathfrak{t}_\mathbf{0} + 115 + 75 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75 + 35 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_1(X_i)$	1	$\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 40 + 30 \cdot \rho_1(X_i) + (\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i) - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_{15}, \mathfrak{t}_{15}$	$\mathfrak{t}_\mathbf{0} + 215 + 75 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75 + 35 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_1(X_i)$	1	$\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 140 + 30 \cdot \rho_1(X_i) + (\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i) - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_{16}, \mathfrak{t}_{16}$	$\mathfrak{t}_\mathbf{0} + 235 + 75 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 75 + 35 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_1(X_i)$	0	$\mathfrak{t}_\mathbf{0} + 215 + 75 \cdot \rho_1(X_i)$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + 28440 + 30 \cdot \rho_1(X_i) + (\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i) - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$
$\Gamma_{17}, \mathfrak{t}_{17}$	$\mathfrak{t}_\mathbf{0} + (235 + 75 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil$	$\mathfrak{e}_\mathbf{0}^\mathbf{cpu} + (75 + 35 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil$	$\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_1(X_i) \cdot \lceil \rho_0(X_n)/10 \rceil$	$\mathbf{on}_\mathbf{0}^\mathbf{r}$	$\tau_\mathbf{0}^\mathbf{r}$	$\mathfrak{e}_\mathbf{0}^\mathbf{r} + (28440 + 30 \cdot \rho_1(X_i) + (\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i) - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})) \cdot \lceil \rho_0(X_n)/10 \rceil$

Table 6. Global states for Algorithm 2.

overestimation only if $\mathbf{on}_\mathbf{0}^\mathbf{r} = 1$. Moreover, after the loop $C_r :: s.on = \mathbf{on}_\mathbf{0}^\mathbf{r}$, which does not lead to further overestimation.

After applying the Hoare logic, we still need to add the time-dependent energy-consumption for each component. The CPU its time-dependent consumption is already amortized in its incidental energy consumptions (although it would be simple to add time-dependent consumption if needed). Its energy consumption at Γ_{17} is thus final. The result for the sensor is $\mathfrak{e}_\mathbf{0}^\mathbf{s} + 40 \cdot \rho_1(X_i) \cdot \lceil \rho_0(X_n)/10 \rceil + (\mathfrak{t}_\mathbf{0} + (235 + 75 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil - \mathfrak{t}_\mathbf{0}) \cdot 3$, which simplifies to $\mathfrak{e}_\mathbf{0}^\mathbf{s} + (705 + 265 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil$. The result for the radio is $\mathfrak{e}_\mathbf{0}^\mathbf{r} + (28440 + 30 \cdot \rho_1(X_i) + (\mathfrak{t}_\mathbf{0} + 75 + 75 \cdot \rho_1(X_i) - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})) \cdot \lceil \rho_0(X_n)/10 \rceil + ((\mathfrak{t}_\mathbf{0} + (235 + 75 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil) - \tau_\mathbf{0}^\mathbf{r}) \cdot (2 + 200 \cdot \mathbf{on}_\mathbf{0}^\mathbf{r})$.

We have not used it before, but we have assumed a pre-analysis which gives us the "standard" Hoare logic pre- and post-conditions. It relates program variables to input values and constants. Such an analysis should easily be able to determine that $\rho_1(X_i) = 10$. We can use this information now to simplify our results.

$$\begin{aligned} C_{cpu}^{final} :: \mathfrak{e} &= \mathfrak{e}_\mathbf{0}^\mathbf{cpu} + (75 + 35 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil \\ &= \mathfrak{e}_\mathbf{0}^\mathbf{cpu} + (75 + 35 \cdot 10) \cdot \lceil \rho_0(X_n)/10 \rceil \\ &= \mathfrak{e}_\mathbf{0}^\mathbf{cpu} + 425 \cdot \lceil \rho_0(X_n)/10 \rceil \end{aligned}$$

$$\begin{aligned} C_{s}^{final} :: \mathfrak{e} &= \mathfrak{e}_\mathbf{0}^\mathbf{s} + (705 + 265 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil \\ &= \mathfrak{e}_\mathbf{0}^\mathbf{s} + (705 + 265 \cdot 10) \cdot \lceil \rho_0(X_n)/10 \rceil \\ &= \mathfrak{e}_\mathbf{0}^\mathbf{s} + 3355 \cdot \lceil \rho_0(X_n)/10 \rceil \end{aligned}$$

$$\begin{aligned}
C_r^{final} :: \mathfrak{e} &= \mathfrak{e}_{\mathbf{0}}^{\mathbf{r}} + (28440 + 30 \cdot \rho_1(X_i) + (\mathfrak{t}_{\mathbf{0}} + 75 + 75 \cdot \rho_1(X_i) - \tau_{\mathbf{0}}^{\mathbf{r}}) \ \cdot \ (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})) \cdot \lceil \rho_0(X_n)/10 \rceil \\
&\quad + ((\mathfrak{t}_{\mathbf{0}} + (235 + 75 \cdot \rho_1(X_i)) \cdot \lceil \rho_0(X_n)/10 \rceil) - \tau_{\mathbf{0}}^{\mathbf{r}}) \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}}) \\
&= \mathfrak{e}_{\mathbf{0}}^{\mathbf{r}} + (28440 + 30 \cdot 10 + (\mathfrak{t}_{\mathbf{0}} + 75 + 75 \cdot 10 - \tau_{\mathbf{0}}^{\mathbf{r}}) \ \cdot \ (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})) \cdot \lceil \rho_0(X_n)/10 \rceil \\
&\quad + ((\mathfrak{t}_{\mathbf{0}} + (235 + 75 \cdot 10) \cdot \lceil \rho_0(X_n)/10 \rceil) - \tau_{\mathbf{0}}^{\mathbf{r}}) \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}}) \\
&= \mathfrak{e}_{\mathbf{0}}^{\mathbf{r}} + (28740 + (\mathfrak{t}_{\mathbf{0}} + 825 - \tau_{\mathbf{0}}^{\mathbf{r}}) \ \cdot \ (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})) \cdot \lceil \rho_0(X_n)/10 \rceil \\
&\quad + ((\mathfrak{t}_{\mathbf{0}} + 985 \cdot \lceil \rho_0(X_n)/10 \rceil) - \tau_{\mathbf{0}}^{\mathbf{r}}) \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}}) \\
&= \mathfrak{e}_{\mathbf{0}}^{\mathbf{r}} + (28740 + (\mathfrak{t}_{\mathbf{0}} + 825 - \tau_{\mathbf{0}}^{\mathbf{r}}) \ \cdot \ (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})) \cdot \lceil \rho_0(X_n)/10 \rceil \\
&\quad + 985 \cdot \lceil \rho_0(X_n)/10 \rceil \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}}) + (\mathfrak{t}_{\mathbf{0}} - \tau_{\mathbf{0}}^{\mathbf{r}}) \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}}) \\
&= \mathfrak{e}_{\mathbf{0}}^{\mathbf{r}} + (28740 + (\mathfrak{t}_{\mathbf{0}} + 1810 - \tau_{\mathbf{0}}^{\mathbf{r}}) \ \cdot \ (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})) \cdot \lceil \rho_0(X_n)/10 \rceil + (\mathfrak{t}_{\mathbf{0}} - \tau_{\mathbf{0}}^{\mathbf{r}}) \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})
\end{aligned}$$

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

$$\begin{gathered}
\mathfrak{e}_{\mathbf{0}}^{\mathbf{cpu}} + 425 \cdot \lceil \rho_0(X_n)/10 \rceil + \mathfrak{e}_{\mathbf{0}}^{\mathbf{s}} + 3355 \cdot \lceil \rho_0(X_n)/10 \rceil \\
+\mathfrak{e}_{\mathbf{0}}^{\mathbf{r}} + (28740 + (\mathfrak{t}_{\mathbf{0}} + 1810 - \tau_{\mathbf{0}}^{\mathbf{r}}) \ \cdot \ (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})) \cdot \lceil \rho_0(X_n)/10 \rceil - (\mathfrak{t}_{\mathbf{0}} - \tau_{\mathbf{0}}^{\mathbf{r}}) \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}}) = \\
\mathfrak{e}_{\mathbf{0}}^{\mathbf{cpu}} + \mathfrak{e}_{\mathbf{0}}^{\mathbf{s}} + \mathfrak{e}_{\mathbf{0}}^{\mathbf{r}} + (32520 + (\mathfrak{t}_{\mathbf{0}} + 1810 - \tau_{\mathbf{0}}^{\mathbf{r}}) \ \cdot \ (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})) \cdot \lceil \rho_0(X_n)/10 \rceil - (\mathfrak{t}_{\mathbf{0}} - \tau_{\mathbf{0}}^{\mathbf{r}}) \cdot (2 + 200 \cdot \mathbf{on}_{\mathbf{0}}^{\mathbf{r}})
\end{gathered}$$

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:

$$\begin{gathered}
(32520 + 1810 \cdot 2) \cdot \lceil \rho_0(X_n)/10 \rceil = \\
36140 \cdot \lceil \rho_0(X_n)/10 \rceil = \\
3614 \cdot \rho_0(X_n)
\end{gathered}$$

5 Soundness of the Logic with Respect to the Semantics

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 $(S), (N), (V)$, thesis follows directly.
- If last rule was $(E), (A)$, thesis follows by induction on the premises. No state is changed in the rule.
- If last rule applied was $(CFinc)$ or $(CStd)$, then thesis follows by Axiom 2.
- If last rule was (LF), thesis follows by induction on the hypothesis.

– If last rule was (C) then we have the following two derivation trees:

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

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 (I), 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 (W). We have the following two derivation trees

$$\frac{\{\mathbf{fix}(\Gamma_1,e;S);\mathfrak{t}_1;\rho_1\}e\{\Gamma_2;\mathfrak{t}_2;\rho_2\} \quad \Gamma_3=\Gamma_2[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_w] \quad \{\Gamma_3;\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_4;\mathfrak{t}_3;\rho_3\}}{\{\Gamma_1;\mathfrak{t}_1;\rho_1\}\mathbf{while}_{rf}\ e\ \mathbf{do}\ S\ \mathbf{end\ while}\{\mathbf{e}(\mathbf{r}(\Gamma_4,\mathfrak{t}_1),\Gamma_1,rf),\mathfrak{t}_1+((\mathfrak{t}_3-\mathfrak{t}_1)*rf);\rho_3\}}(\mathrm{W})$$

$$\frac{\{\mathbf{fix}(\Gamma_6,e;S);\mathfrak{t}_6;\rho_1\}e\{\Gamma_7;\mathfrak{t}_7;\rho_2\} \quad \Gamma_8=\Gamma_7[C_{cpu}::\mathfrak{e}\mathrel{+}=C_{cpu}::\mathfrak{E}_w] \quad \{\Gamma_8;\mathfrak{t}_7+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_9;\mathfrak{t}_8;\rho_3\}}{\{\Gamma_6;\mathfrak{t}_6;\rho_1\}\mathbf{while}_{rf}\ e\ \mathbf{do}\ S\ \mathbf{end\ while}\{\mathbf{e}(\mathbf{r}(\Gamma_9,\mathfrak{t}_6),\Gamma_6,rf),\mathfrak{t}_6+((\mathfrak{t}_8-\mathfrak{t}_6)*rf);\rho_3\}}(\mathrm{W})$$

First we apply the induction hypothesis on both $\{\mathbf{fix}(\Gamma_1,e;S);\mathfrak{t}_1;\rho_1\}e\{\Gamma_2;\mathfrak{t}_2;\rho_2\}$ and on $\{\mathbf{fix}(\Gamma_6,e;S);\mathfrak{t}_6;\rho_1\}e\{\Gamma_7;\mathfrak{t}_7;\rho_2\}$. Notice that, by the definition of the fixpoint function, if $\Gamma_6 \geq \Gamma_1$ then $\mathbf{fix}(\Gamma_6,e;S) \geq \mathbf{fix}(\Gamma_1,e;S)$. Hence this ensures us that for every i, if $C_i^{\Gamma_6} \geq C_i^{\Gamma_1}$ then $C_i^{\Gamma_7} \geq C_i^{\Gamma_2}$.
Since the next Hoare rule does not change the state, we can apply the induction hypothesis between $\{\Gamma_3;\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_4;\mathfrak{t}_3;\rho_3\}$ and $\{\Gamma_8;\mathfrak{t}_7+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_9;\mathfrak{t}_8;\rho_3\}$.
By concatenating all the implications we have that for every i, if $C_i^{\Gamma_6} \geq C_i^{\Gamma_1}$ then $C_i^{\Gamma_9} \geq C_i^{\Gamma_4}$.
Since the other two functions (revert and energy usage) don't change the state, the thesis is proved.

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. □

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. □

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.

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 $\langle S,\sigma',\Gamma',\mathfrak{t}'\rangle\Downarrow^s\langle\sigma'',\Gamma'',\mathfrak{t}''\rangle$, 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 (N) or (V) or (S) then thesis holds.
– If last rule was (E), then we have these two derivations:

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

$$\frac{\dfrac{\langle e_1,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle n,\sigma',\Gamma',\mathfrak{t}'\rangle \qquad C_{cpu}::\boxdot(e_1,e_2)=p}{\langle e_2,\sigma',\Gamma',\mathfrak{t}'\rangle\Downarrow^e\langle m,\sigma'',\Gamma'',\mathfrak{t}''\rangle \quad \Gamma'''=\Gamma''[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_e]}(\mathrm{sE})}{\dfrac{\langle e_1\boxdot e_2,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle p,\sigma'',\Gamma''',\mathfrak{t}''+C_{cpu}::\mathfrak{T}_e\rangle}{\langle e_1\boxdot e_2,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''+C_{cpu}::\mathfrak{T}_e\rangle}}$$

By applying induction hypothesis we easily retrieve the thesis, indeed same time is added on both derivations.
– If last rule was (A), as in the last case, by applying induction hypothesis we get the thesis.
– If last rule was $(CFinc)$ or $(CFtd)$, property holds, since we are adding the same amount of time on both derivations.
– If last rule was (LF), then by applying induction on the premise thesis follows.
– If last rule was (C), thesis follows by applying induction hypothesis on the premises.
– If last rule was (I), 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 (W). Concerning the semantics derivation, we could have applied (sWF) or (sWT). Clearly the interesting case is when the last semantic rule was (sWT). We have the following two derivations:

$$\frac{\dfrac{\{\mathbf{fix}(\Gamma_1,e;S);\mathfrak{t}_1;\rho_1\}e\{\Gamma_2;\mathfrak{t}_2;\rho_2\} \quad \{\Gamma_3;\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_4;\mathfrak{t}_3;\rho_3\}}{\{\Gamma_1;\mathfrak{t}_1;\rho_1\}\mathbf{while}_{rf}\ e\ \mathbf{do}\ S\ \mathbf{end\ while}\{\mathbf{e}(\mathbf{r}(\Gamma_4,\mathfrak{t}_1),\Gamma_1,rf),\mathfrak{t}_1+((\mathfrak{t}_3-\mathfrak{t}_1)*rf);\rho_3\}}(\mathrm{W})}{\dfrac{\Gamma''=\Gamma'[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_w] \quad \dfrac{\langle e,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle n,\sigma',\Gamma',\mathfrak{t}'\rangle \qquad \langle S_1,\sigma',\Gamma'',\mathfrak{t}'+C_{cpu}::\mathfrak{T}_w\rangle\Downarrow^s\langle\sigma''',\Gamma'''',\mathfrak{t}''''\rangle \quad \langle\mathbf{while}_{rf}\ e\ \mathbf{do}\ S_1\ \mathbf{end\ while},\sigma''',\Gamma'''',\mathfrak{t}''''\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''\rangle}{\langle S_1;\mathbf{while}_{rf}\ e\ \mathbf{do}\ S_1\ \mathbf{end\ while},\sigma',\Gamma'',\mathfrak{t}'+C_{cpu}::\mathfrak{T}_w\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''\rangle \quad n\neq 0}(\mathrm{sC})}{\langle\mathbf{while}_{rf}\ e\ \mathbf{do}\ S_1\ \mathbf{end\ while},\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''\rangle}(\mathrm{sWT})}$$

We apply induction hypothesis on $\{\mathbf{fix}(\Gamma_1,e;S);\mathfrak{t}_1;\rho_1\}e\{\Gamma_2;\mathfrak{t}_2;\rho_2\}$ 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 $\{\Gamma_3;\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_4;\mathfrak{t}_3;\rho_3\}$ and on $\langle S_1,\sigma',\Gamma'',\mathfrak{t}'+C_{cpu}::\mathfrak{T}_w\rangle\Downarrow^s\langle\sigma''',\Gamma'''',\mathfrak{t}''''\rangle$. Clearly we may apply the induction hypothesis on all the sub-trees of $\langle\mathbf{while}_{rf}\ e\ \mathbf{do}\ S_1\ \mathbf{end\ while},\sigma''',\Gamma'''',\mathfrak{t}''''\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''\rangle$ in the semantic tree. Indeed, the induction says that this holds for every states $\Gamma;\mathfrak{t}$, which means that $\mathfrak{t}_3-\mathfrak{t}_1$ is a good overestimation for every single possible cycles of the loop. Since rf is an overestimation of the loop, it follows that clerly $\mathfrak{t}_1+(\mathfrak{t}_3-\mathfrak{t}_1)*rf\geq\mathfrak{t}''-\mathfrak{t}$ 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 $(S),(N),(V)$, thesis follows directly. Nothing changes
– If last rule was $(E),(A)$, 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 $(CFinc)$ or $(CFtd)$, 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 (LF), thesis follows by induction on the hypothesis.

– If last rule was (C) then we have the following two derivation trees:

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

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 (I), 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.

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

$$\frac{\{\mathbf{fix}(\Gamma_1,e;S);\mathfrak{t}_1;\rho_1\}e\{\Gamma_2;\mathfrak{t}_2;\rho_2\} \quad \{\Gamma_3;\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_4;\mathfrak{t}_3;\rho_3\}}{\{\Gamma_1;\mathfrak{t}_1;\rho_1\}\mathbf{while}_{rf}\ e\ \mathbf{do}\ S\ \mathbf{end\ while}\{\mathbf{e}(\mathbf{r}(\Gamma_4,\mathfrak{t}_1),\Gamma_1,rf),\mathfrak{t}_1+((\mathfrak{t}_3-\mathfrak{t}_1)*rf);\rho_3\}}(\mathrm{W}) \quad \Gamma_3=\Gamma_2[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_w]$$

$$\frac{\{\mathbf{fix}(\Gamma_6,e;S);\mathfrak{t}_6;\rho_1\}e\{\Gamma_7;\mathfrak{t}_7;\rho_2\} \quad \{\Gamma_8;\mathfrak{t}_7+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_9;\mathfrak{t}_8;\rho_3\}}{\{\Gamma_6;\mathfrak{t}_6;\rho_1\}\mathbf{while}_{rf}\ e\ \mathbf{do}\ S\ \mathbf{end\ while}\{\mathbf{e}(\mathbf{r}(\Gamma_9,\mathfrak{t}_6),\Gamma_6,rf),\mathfrak{t}_6+((\mathfrak{t}_8-\mathfrak{t}_6)*rf);\rho_3\}}(\mathrm{W}) \quad \Gamma_8=\Gamma_7[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_w]$$

We assume, so, that $(\Gamma_6;\mathfrak{t}_6)\geq(\Gamma_1;\mathfrak{t}_1)$. Hence the correspondent fixpoints preserve the ordering, by their definition. Indeed the fixpoint is a concatenation of δ function that preserve the order of states. Since they preserve the ordering by Axiom 2 and the time analysis is independent (is always the same by theorem 3), the ordering is preserved. First we apply induction hypothesis on $\{\mathbf{fix}(\Gamma_1,e;S);\mathfrak{t}_1;\rho_1\}e\{\Gamma_2;\mathfrak{t}_2;\rho_2\}$ and on $\{\mathbf{fix}(\Gamma_6,e;S);\mathfrak{t}_6;\rho_1\}e\{\Gamma_7;\mathfrak{t}_7;\rho_2\}$. This ensure us that $(\Gamma_7;\mathfrak{t}_7)\geq(\Gamma_2;\mathfrak{t}_2)$. 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 $(\Gamma_8;\mathfrak{t}_7+C_{cpu}::\mathfrak{T}_w;\rho_2)\geq(\Gamma_3;\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_w;\rho_2)$. This means that we can apply induction hypothesis between $\{\Gamma_3;\mathfrak{t}_2+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_4;\mathfrak{t}_4;\rho_3\}$ and $\{\Gamma_8;\mathfrak{t}_7+C_{cpu}::\mathfrak{T}_w;\rho_2\}S\{\Gamma_9;\mathfrak{t}_9;\rho_3\}$. By concatenating the implications we have that $(\Gamma_9;\mathfrak{t}_9)\geq(\Gamma_4;\mathfrak{t}_4)$. Finally we have to regress the state. Since the analysis computes on the same slice of time, as proven in Lemma 3, and since $(\Gamma_9;\mathfrak{t}_9)\geq(\Gamma_4;\mathfrak{t}_4)$ it follows that
$(\mathbf{e}(\mathbf{r}(\Gamma_{11},\mathfrak{t}_6),\Gamma_9,\Gamma_6)rf,\mathfrak{t}_6+((\mathfrak{t}_9-\mathfrak{t}_6)*rf);\rho_3)\geq(\mathbf{e}(\mathbf{r}(\Gamma_5',\mathfrak{t}_1),\Gamma_4,\Gamma_1)rf,\mathfrak{t}_1+((\mathfrak{t}_4-\mathfrak{t}_1)*rf);\rho_3)$

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* $\langle S,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^s\langle\sigma',\Gamma',\mathfrak{t}'\rangle$ *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* $\langle e,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle e',\sigma',\Gamma',\mathfrak{t}'\rangle$ *then for every device* i, $C_i^{\Gamma_1}::s\geq C_i^{\Gamma'}::s$.

Proof. Proof is proven by structural induction on the derivation both for semantics and for analysis. 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 (N) or (V), thesis holds since no component has been touched.

– If last rule in the analysis was (E) we have these two derivations:

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

$$\frac{\langle e_1,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle n,\sigma',\Gamma',\mathfrak{t}'\rangle \quad C_{cpu}::\boxdot(e_1,e_2)=p \quad \langle e_2,\sigma',\Gamma',\mathfrak{t}'\rangle\Downarrow^e\langle m,\sigma'',\Gamma'',\mathfrak{t}''\rangle \quad \Gamma'''=\Gamma''[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_e]}{\langle e_1\boxdot e_2,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle p,\sigma'',\Gamma''',\mathfrak{t}''+C_{cpu}::\mathfrak{T}_e\rangle}(\mathrm{sE})$$

We can apply the induction hypothesis on $\{\Gamma;\mathfrak{t};\rho\}e_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\}$ and $\langle e_1,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle n,\sigma',\Gamma',\mathfrak{t}'\rangle$. 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 $\langle e_2,\sigma',\Gamma',\mathfrak{t}'\rangle\Downarrow^e\langle m,\sigma'',\Gamma'',\mathfrak{t}''\rangle$, 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 (A), then the last semantic rule was (sA). Thesis is proven by induction on the premise.
- If last production rule was $(CFtd)$, then the last semantic rule was (sCF). We are applying the same $C_i::\delta_f()$ on both side.
- If last production rule was $(CFinc)$, then the last semantic rule was (sCF). We are applying the same $C_i::\delta_f()$ on both side.
- If last production rule was (LF), then the last semantic rule was (sLF). Thesis is proven by induction on the premise.
- If last semantic rule was $(sEasS)$, then whatever is the production rule, thesis is proven by induction on the premise of the semantic tree.
- If last production rule was (S), then the last semantic rule was (sS). Thesis holds.
- If last production rule was (C), then the last semantic rule was (sC). We have these two derivation trees:

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

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

We can cleary apply induction hypothesis on $\{\Gamma;\mathfrak{t};\rho\}S_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\}$ and $\langle S_1,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^s\langle\sigma',\Gamma',\mathfrak{t}'\rangle$ 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 $\langle S_2,\sigma',\Gamma',\mathfrak{t}'\rangle\Downarrow^s\langle\sigma'',\Gamma',\mathfrak{t}'\rangle$ 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 (I), 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:

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

$$\frac{\begin{array}{cc}\langle e,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle 0,\sigma',\Gamma',\mathfrak{t}'\rangle & \langle S_1,\sigma',\Gamma'',\mathfrak{t}'+C_{cpu}::\mathfrak{T}_{ite}\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''\rangle\\ & \Gamma''=\Gamma'[C_{cpu}::\mathfrak{c}\ +\!\!=C_{cpu}::\mathfrak{E}_{ite}]\end{array}}{\langle\textbf{if }e\textbf{ then }S_1\textbf{ else }S_2\textbf{ end if},\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''\rangle}(\text{sIF})$$

We can apply induction hypothesis on $\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\}$ and $\langle e,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle 0,\sigma',\Gamma',\mathfrak{t}'\rangle$ 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 $\{\Gamma'';\mathfrak{t}'+C_{cpu}::\mathfrak{T}_{ite};\rho''\}S_1\{\Gamma_6;\mathfrak{t}_6;\rho_6\}$. 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 (W), we could have two possible last semantic ruled applied. We consider the case where the last semantic rule was (sWT); in the other case, thesis follows straightforward. We have the following derivation trees:

$$\frac{\begin{array}{cc}\{\textbf{fix}(\Gamma,e;S);\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} & \Gamma_2=\Gamma_1[C_{cpu}::\mathfrak{c}\ +\!\!=C_{cpu}::\mathfrak{E}_w]\\ & \{\Gamma_2;\mathfrak{t}_1+C_{cpu}::\mathfrak{T}_w;\rho_1\}S\{\Gamma_3;\mathfrak{t}_2;\rho_2\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\textbf{while}_{rf}\ e\ \textbf{do}\ S\ \textbf{end while}\{\mathfrak{e}(\mathbf{r}(\Gamma_3,\mathfrak{t}),\Gamma,rf),\mathfrak{t}+((\mathfrak{t}_2-\mathfrak{t})*rf);\rho_3\}}(\text{W})$$

$$\frac{\Gamma'' = \Gamma'[C_{cpu} :: \mathfrak{e} \mathrel{+}= C_{cpu} :: \mathfrak{E}_w] \quad \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \dfrac{\langle S_1, \sigma', \Gamma'', \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w\rangle \Downarrow^s \langle \sigma''', \Gamma'''', \mathfrak{t}'''\rangle \quad \langle \textbf{while}_{rf}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma''', \Gamma'''', \mathfrak{t}'''\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle}{\langle S_1; \textbf{while}_{rf}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma', \Gamma'', \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle}\text{sC} \quad n \neq 0}{\langle \textbf{while}_{rf}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle}(\text{sWT})$$

We can apply induction hypothesis on $\{\textbf{fix}(\Gamma, e; S); \mathfrak{t}; \rho\}e\{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ and $\langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle$ and 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 can apply induction hypothesis between $\{\Gamma''; \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w; \rho_1\}S\{\Gamma_3'; \mathfrak{t}_2'; \rho_2'\}$ $\langle S_1, \sigma', \Gamma'', \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w\rangle \Downarrow^s \langle \sigma''', \Gamma'''', \mathfrak{t}'''\rangle$. By lemma 2 we get that for every i, $C_i^{\Gamma_3} \geq C_i^{\Gamma''''}$.
By definition, notice that $\textbf{fix}(\Gamma_3; e; S)$ is greater than all the component states that could be found at the beginning of each cycle in $\langle S_1; \textbf{while}_{rf}\ e\ \textbf{do}\ S_1\ \textbf{end while}, \sigma', \Gamma''', \mathfrak{t}'''\rangle \Downarrow^s \langle \sigma'', \Gamma'''', \mathfrak{t}''''\rangle$. By the theorem 2 we can easily conclude that the states in $\textbf{fix}(\Gamma_3; e; S)$ are equal to the states found in Γ_3. Hence $\Gamma_3 \geq \Gamma'''$.

This concludes the proof. □

Theorem 6 (soundness). *For every global timing* $\mathfrak{t}$ *and for every set of component states* Γ*, if* $\langle S_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma', \Gamma', \mathfrak{t}'\rangle$ *and* $\{\mathfrak{t}; \Gamma; \rho\}S_1\{\mathfrak{t}_1; \Gamma_1; \rho_1\}$ *or if* $\langle e_1, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle \sigma', \Gamma', \mathfrak{t}'\rangle$ *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 structural induction on the semantic and production rule derivations.

- If last semantic rule was (sN) or (sV), thesis holds, since nothing has been changed.
- If last semantic rule was (sE), last production rule was surely (E). We have the following case:

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

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

We apply induction hypothesis on $\{\Gamma; \mathfrak{t}; \rho\}e_1\{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ and $\{\Gamma; \mathfrak{t}; \rho\}e_1\{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ 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 $\langle e_2, \sigma', \Gamma', \mathfrak{t}'\rangle \Downarrow^e \langle m, \sigma'', \Gamma'', \mathfrak{t}''\rangle$. 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 (sA), then the last production rule was surely (A). We have the following case:

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

$$\frac{\langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \Gamma'' = \Gamma'[C_{cpu} :: \mathfrak{e} \mathrel{+}= C_{cpu} :: \mathfrak{E}_a]}{\langle X_1 = e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma[n/X_1], \Gamma'', \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_a\rangle}(\text{sA})$$

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 (sCF), then we could have two possible production rules. Actually if the state doesn't change, the last production rule is $(CFinc)$ and since we are just adding the same amount of time and energy on both side, the thesis holds. In case the last production rule was $(CFtd)$, we have the following case:

$$\frac{\Gamma_1 = \Gamma[C_i :: s \leftarrow C_i :: \delta_f(C_i :: s), C_i :: \tau \leftarrow \mathfrak{t}, C_i :: \mathfrak{e} \mathrel{+}= C_i :: \mathfrak{E}_f + td(C_i, \mathfrak{t})]}{\{\Gamma; \mathfrak{t}; \rho\}C_i :: f(args)\{\Gamma_1; \mathfrak{t}_1[X_j, \ldots, X_k/\rho(X_j, \ldots, X_k)] + C_i :: \mathfrak{T}_f; \rho\}}(\text{CFtd})$$

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

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 (sLF), clearly last production rule was (FL). We have the following case:

$$\frac{\{\Gamma;\mathfrak{t};\rho\}S\{\Gamma_1;\mathfrak{t}_1;\rho_1\} \quad \textbf{function } name\,(X_1,\ldots,X_n)\ S \textbf{ return } X; \textbf{ end function} \in \textit{def\_functions}}{\{\Gamma;\mathfrak{t};\rho\}name(X_j,\ldots,X_{j+n})\{\Gamma_1;\mathfrak{t}_1[X_1,\ldots,X_n/\rho(X_j,\ldots,X_{j+n})];\rho\}}(\text{LF})$$

$$\frac{\begin{array}{c}\langle S,\sigma^0[n_1,n_2,\ldots/X_1,X_2,\ldots],\Gamma,\mathfrak{t}\rangle\Downarrow^s\langle\sigma^1,\Gamma^1,\mathfrak{t}^1\rangle \quad \sigma^0 \text{ fresh}\\ \textbf{function } name\,(X_1,X_2\ldots)\ S \textbf{ return } X_i; \textbf{ end function} \in \textit{def\_functions}\end{array}}{\langle name(n_1,n_2\ldots),\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle\sigma'(X_i),\sigma,\Gamma',\mathfrak{t}'\rangle}(\text{sLF})$$

Thesis holds by induction on the premise. Clearly we have to make substitution with actual values on the analysis, otherwise the result cold not be true; The derivation in the semantics gives the result with actual value, hence we have to do the same in the production derivation.

– If last semantic rule was $(sEasS)$, 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 (sS), then last production rule was (S). Thesis holds, since nothing has been changed.

– If last semantic rule was (sC), then last production rule was (S). We have the following case:

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

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

We apply induction hypothesis on $\{\Gamma;\mathfrak{t};\rho\}S_1\{\Gamma_1;\mathfrak{t}_1;\rho_1\}$ and $\langle S_1,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^s\langle\sigma',\Gamma',\mathfrak{t}'\rangle$. 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 $\langle S_2,\sigma',\Gamma',\mathfrak{t}'\rangle\Downarrow^s\langle\sigma'',\Gamma'',\mathfrak{t}''\rangle$. We obtain that $(\Gamma_2';\mathfrak{t}_2')\geq(\Gamma'';\mathfrak{t}'')$. By lemma 4 we conclude.

– If last semantic rule was (sIF), then last production rule was (I). We have the following case:

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

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

Clearly we can apply induction hypothesis on $\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\}$ and $\langle e,\sigma,\Gamma,\mathfrak{t}\rangle\Downarrow^e\langle 0,\sigma',\Gamma',\mathfrak{t}'\rangle$. 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 $(\Gamma_2;\mathfrak{t}_1+C_{cpu}::\mathfrak{T}_{ite};\rho_1)\geq(\Gamma'';\mathfrak{t}'+C_{cpu}::\mathfrak{T}_{ite})$. Just for the purpose of proof, we derive the following new analysis $\{\Gamma'';\mathfrak{t}'+C_{cpu}::\mathfrak{T}_{ite};\rho_1\}S_1\{\Gamma_3';\mathfrak{t}_3';\rho_3\}$ and we apply the induction hypothesis with $\langle S_1,\sigma',\Gamma'',\mathfrak{t}'+C_{cpu}::\mathfrak{T}_{ite}\rangle\Downarrow^s\langle\sigma'',\Gamma''',\mathfrak{t}''\rangle$. 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 (sIT). This case is similar to the last case, where we analysed (sIF).

– If last semantic rule was (sWF). Clearly last production rule was (W). Thesis surely holds, since in the semantic rule nothing has been touched.

– If last semantic rule was (sWT). Last production rule was (W). We have the following case:

$$\frac{\begin{array}{ccc}\{\Gamma;\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} & & \\ \Gamma_2=\Gamma_1[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_w] & & \Gamma_5=\Gamma_4[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_w]\\ \{\Gamma_2;\mathfrak{t}_1+C_{cpu}::\mathfrak{T}_w;\rho_1\}S\{\Gamma_3;\mathfrak{t}_2;\rho_2\} & \{\textbf{fix}(\Gamma_3;e;S);\mathfrak{t}_2;\rho_1\}e;S\{\Gamma_5;\mathfrak{t}_4+C_{cpu}::\mathfrak{T}_w;\rho_2\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\textbf{while}_{rf}\ e \textbf{ do } S \textbf{ end while}\{\textbf{e}(\textbf{r}(\Gamma_5,\mathfrak{t}),\Gamma_3,\Gamma)rf,\mathfrak{t}+((\mathfrak{t}_2-\mathfrak{t})*rf);\rho_3\}}(\text{W})$$

$$\frac{\begin{array}{cc}\{\textbf{fix}(\Gamma,e;S);\mathfrak{t};\rho\}e\{\Gamma_1;\mathfrak{t}_1;\rho_1\} & \Gamma_2=\Gamma_1[C_{cpu}::\mathfrak{e}\ +\!\!=C_{cpu}::\mathfrak{E}_w]\\ & \{\Gamma_2;\mathfrak{t}_1+C_{cpu}::\mathfrak{T}_w;\rho_1\}S\{\Gamma_3;\mathfrak{t}_2;\rho_2\}\end{array}}{\{\Gamma;\mathfrak{t};\rho\}\textbf{while}_{rf}\ e \textbf{ do } S \textbf{ end while}\{\textbf{e}(\textbf{r}(\Gamma_3,\mathfrak{t}),\Gamma,rf),\mathfrak{t}+((\mathfrak{t}_2-\mathfrak{t})*rf);\rho_3\}}(\text{W})$$

$$\frac{\Gamma'' = \Gamma'[C_{cpu} :: \mathfrak{e} \mathrel{+}= C_{cpu} :: \mathfrak{E}_w] \quad \langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle \quad \langle S_1; \mathbf{while}_{rf}\ e\ \mathbf{do}\ S_1\ \mathbf{end\ while}, \sigma', \Gamma'', \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle \quad n \neq 0}{\langle \mathbf{while}_{rf}\ e\ \mathbf{do}\ S_1\ \mathbf{end\ while}, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle} (\text{sWT})$$

We can apply induction hypothesis on $\{\mathbf{fix}(\Gamma, e; S); \mathfrak{t}; \rho\} e \{\Gamma_1; \mathfrak{t}_1; \rho_1\}$ and $\langle e, \sigma, \Gamma, \mathfrak{t}\rangle \Downarrow^e \langle n, \sigma', \Gamma', \mathfrak{t}'\rangle$. We retrieve that $(\Gamma_1; \mathfrak{t}_1) \geq (\Gamma'; \mathfrak{t}')$, moreover, by theorem 5 we retrieve that $\Gamma_1 \geq \Gamma'$.
Since we are not modifying any state and just add same amount of energy and timing, we derive that $(\Gamma_2; \mathfrak{t}_1 + C_{cpu} :: \mathfrak{T}_w) \geq (\Gamma''; \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w)$ and $\Gamma_2 \geq \Gamma''$.
Just for the purpose of proof we create the new derivation $\{\Gamma''; \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w; \rho_1\} S \{\Gamma_3'; \mathfrak{t}_3'; \rho_2\}$ and we apply induction hypothesis with $\langle S_1; \mathbf{while}_{rf}\ e\ \mathbf{do}\ S_1\ \mathbf{end\ while}, \sigma', \Gamma'', \mathfrak{t}' + C_{cpu} :: \mathfrak{T}_w\rangle \Downarrow^s \langle \sigma'', \Gamma''', \mathfrak{t}''\rangle$. 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 $(\Gamma_3; \mathfrak{t}_2) \geq (\Gamma'''; \mathfrak{t}'')$ and by lemma 2 we get also that $\Gamma_3 \geq \Gamma'''$.
Notice that the fixpoint we calculated at the beginning retrieve a component state that is bigger than every component state at beginning 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 $\mathbf{fix}(\Gamma_3; e; S)$ can be taken as an overbound for the all the cycles. (Remember that $C_i^{\mathbf{fix}(\Gamma; e; S)} :: s = C_i^{\Gamma_3} :: s$)
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 ranking function rf expresses that) plus the possible energy not take in account at the end of the loop.
Formally:

$$\begin{aligned}(C_i^{\mathbf{fix}(\Gamma_3; e; S)} :: \mathfrak{e} - C_i^{\Gamma} :: \mathfrak{e}) + td(C_i^{\Gamma_3}, \mathfrak{t}_2)) * (rf) + C_i^{\Gamma} :: \mathfrak{e} = \\ (C_i^{\Gamma_3} :: \mathfrak{e} - C_i^{\Gamma} :: \mathfrak{e}) + td(C_i^{\Gamma_3}, \mathfrak{t}_2)) * (rf) + C_i^{\Gamma} :: \mathfrak{e}\end{aligned}$$

The results seems similar to the energy computed but it is not. Indeed, the energy computed by the production rules lacks of $td(C_i^{\Gamma_3}, \mathfrak{t}_2) * (rf)$.
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 $td(C_i^{\Gamma_3}, \mathfrak{t}_2) * (rf)$ (the sum of all the little delta not calculated in the energy). Hence $(\mathbf{e}(\mathbf{r}(\Gamma_3, \mathfrak{t}), \Gamma, rf); \mathfrak{t} + ((\mathfrak{t}_2 - \mathfrak{t}) * rf)) \geq (\Gamma'''; \mathfrak{t}''')$.

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 estimating the energy-consumption of software statically in a way which is parametric with respect to hardware models. This is a first step into a hybrid approach to energy consumption analysis in which the software is analysed automatically together with the hardware it controls. Such an automatic static energy-analysis would represent a big step forward, as at this moment, programmers are mostly unaware of the consequences of their choices with respect to energy-consumption.

Future Work We have taken a first step into the development of a general analysis of energy consumption of hybrid systems in which software controls the energy consumption of hardware components. Many future research directions can be envisioned: e.g. 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.

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. Raghunathan, V., Schurgers, C., Park, S., Srivastava, M., Shaw, B.: Energy-aware wireless microsensor networks. In: IEEE Signal Processing Magazine. (2002) 40–50
5. 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
6. 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
7. Cohen, M., Zhu, H.S., Senem, E.E., Liu, Y.D.: Energy types. SIGPLAN Not. **47**(10) (October 2012) 831–850
8. Baek, W., Chilimbi, T.M.: Green: a framework for supporting energy-conscious programming using controlled approximation. SIGPLAN Not. **45**(6) (June 2010) 198–209
9. 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
10. 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)
11. Gheorghita, S.V., Corporaal, H., Basten, T.: Iterative compilation for energy reduction. J. Embedded Comput. **1**(4) (2005) 509–520
12. 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
13. 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
14. 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
15. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In Steffen, B., Levi, G., eds.: Verification, Model Checking, and Abstract Interpretation. Volume 2937 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2004) 465–486
16. 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)
17. 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
18. Shkaravska, O., van Eekelen, M.C.J.D., van Kesteren, R.: Polynomial size analysis of first-order shapely functions. Logical Methods in Computer Science **5**(2) (2009) 1–35 Special Issue with Selected Papers from TLCA 2007.
19. Shkaravska, O., Kersten, R., Van Eekelen, M.: Test-based inference of polynomial loop-bound functions. In Krall, A., Mössenböck, H., eds.: PPPJ'10: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java. ACM Digital Proceedings Series (2010) 99–108