Invariants and Robustness of BIP Models

(Report on Ongoing Work)

Jan Olaf Blech, Thanh-Hung Nguyen, Michaël Périn
Verimag Laboratory, Université de Grenoble, France

Abstract

Verification techniques have become popular in software and hardware development. They increase confidence and potentially provide rich feedback. However, with increasing complexity verification techniques are more likely to contain errors themselves. Many verification tools use invariants of the considered systems for their analysis. These invariants are often generated by the verification tools in a first step. The correctness of these invariants is crucial for the analysis results.

In this paper we present on-going work addressing the problem of automatically generating realistic and guaranteed correct invariants. Since invariant generation mechanisms are error-prone, after the computation of invariants by a verification tool, we formally prove that the generated invariants are indeed invariants of the considered systems using a higher-order theorem prover and automated techniques. We regard invariants for BIP models. BIP (behavior, interaction, priority) is a language for specifying asynchronous component based systems. Proving that an invariant holds often requires an induction on possible system execution traces. For this reason, apart from generating invariants that precisely capture a system's behavior, inductiveness of invariants is an important goal.

We establish a notion of robust BIP models. These can be automatically constructed from our original non-robust BIP models and over-approximate their behavior. We motivate that invariants of robust BIP models capture the behavior of systems in a more natural way than invariants of corresponding non-robust BIP models. Robust BIP models take imprecision due to values delivered by sensors into account. Invariants of robust BIP models tend to be inductive and are also invariants of the original non-robust BIP model. Therefore they may be used by our verification tools and it is easy to show their correctness in a higher-order theorem prover.

The presented work is developed to verify the results of a deadlock-checking tool for embedded systems after their computations. Therewith, we gain confidence in the provided analysis results.

1 Introduction

Verification tools to ensure properties of complex systems have become popular in many application areas. One major goal is to guarantee safety and security properties of the considered systems. These can be computed by generating invariants of the considered systems in a first step and analyzing them. However, as verification tools become more and more complex it is not always easy to see if they are themselves working correctly. An incorrect verification tool might state a wrong property about a system.

Guided by first experiments on automatically verifying – thereby establishing a correctness certificate – the results of a verification tool after they have been computed in a higher-order theorem prover (Coq), we verify invariants of given systems (Bip models) within a theorem prover, we introduce a notion of robustness for these systems. The invariants that are subject to this paper are computed and used by the D-Finder [BBSN08] tool that decides deadlock-freedom of systems modeled in the Bip language [BBS06]. The Bip language features the descriptive power to model asynchronous systems and is designed for building real-time embedded systems consisting of heterogeneous components. Invariants that are both inductive and capture the behavior of our systems in an adequate way are highly desirable for our analysis and verification tools.

In our case study, we require invariants to be inductive in order to be verified automatically by deductive methods. We motivate a technique that is likely to produce inductive invariants: we establish a notion of robust Bip models. These models take imprecision of values due to physical measurements into account. Invariants of robust systems are aimed at describing a system's behavior in a realistic way taking sensor sensitivity and imprecision into account while preserving the necessary precision to be used as basis for analysis results. Our invariants are suitable for automated verification using a discrete semantics. We present a mapping from non-robust to robust systems and prove that invariants of robust systems are also invariants of the original non-robust systems. This allows us to reuse invariant based analysis results for these systems.

1.1 Our Case Study: Guaranteeing Correctness of the Results of a Verification Tool

Robust Bip models are used to make the process (called certification) of automatically proving the results of a deadlock-detection verification tool easier thereby guaranteeing the correctness of its verdict. The overall approach of this tool and the verification process guaranteeing that its results are correct is described in the following two paragraphs.

1.1.1 The Deadlock-detection Tool D-Finder and the Certification of its Results

The deadlock-detection tool D-Finder takes Bip models as inputs and decides whether they are deadlock free. In order to do this, in a first step invariants of these are computed. These are analyzed for potential deadlocks. D-Finder is aimed to be safe in a way that it might detect false positives, i.e., potential deadlocks that do not exist. The computation of invariants is the most sophisticated step within D-Finder. In addition to D-Finder's algorithms an external tool Omega [Ome00] is used in the invariant generation process to perform quantifier elimination. In a second step these invariants are checked to be deadlock-free by using the external SMT solver Yices [DM06] and a definition of deadlock-states.

Verifying that invariants hold is used for guaranteeing the absence of deadlocks in our guiding case study [BP08, BP08a]. The methodology underlying this case study is depicted in Figure 1. Bip models are passed to D-Finder, the deadlock-detection tool. In this paper, we do not trust D-Finder in a first place, but want to establish proofs, that it has indeed worked correctly for each run of this tool. Apart from detecting deadlocks, a certificate is generated by some part of the tool (denoted CertGen). This certificate comprises a proof of deadlock-freedom and is passed to a theorem prover. The D-Finder tool computes invariants and uses them to decide whether a system is deadlock-free or not. Most important to this paper is the fact, that the certificates contain these invariants as well as a proof script that is generated by the certificate generator proving that the invariants do indeed hold. The theorem prover uses this proof script to prove that a Bip model is indeed deadlock-free.

1.1.2 Proving Deadlock-freedom

To verify that a system is indeed deadlock-free in the theorem prover, we have to check the certificates. We break this task of verifying deadlock-freedom for a given Bip model BM down into different subtasks as shown in Figure 2. The proofs for these subtasks are composed to

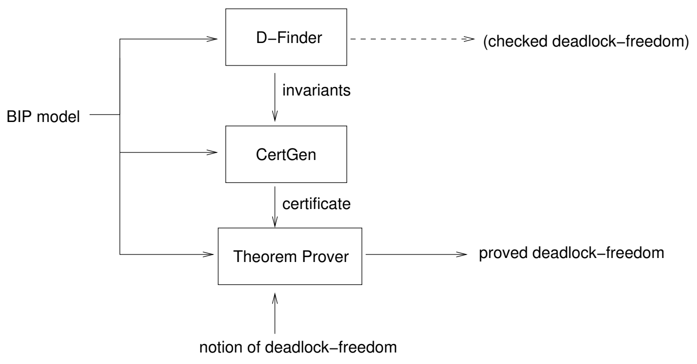

Figure 1: Our Methodology

1. $\forall s.\mathit{ReachableStates}_{BM}(s) \longrightarrow \mathit{Enabled}_{BM}(s)$

$\uparrow$ *transitivity*

2. $\forall s.\mathit{ReachableStates}_{BM}(s) \longrightarrow \neg \mathit{DIS}_{BM}(s)$ *and* [PO1] $\forall s.\neg \mathit{DIS}_{BM}(s) \longrightarrow \mathit{Enabled}_{BM}(s)$

$\uparrow$ *transitivity*

3. [PO2] $\forall s.\mathit{ReachableStates}_{BM}(s) \longrightarrow \Psi_{BM}(s)$ *and* [PO3] $\forall s.\Psi_{BM}(s) \longrightarrow \neg \mathit{DIS}_{BM}(s)$

Figure 2: Verifying Deadlock-freedom: The Meta-Proof

prove the top line. In the figure, we use the following definition of enabled states capturing Bip states from which a state transition to a succeeding state is possible:

$$\mathit{Enabled}_{BM}(s) \equiv \exists s'.(s, s') \in [\![BM]\!]_{BIP} \wedge s \neq s'$$

The $[\![BM]\!]_{BIP}$ denotes the set of possible state transitions of the Bip model BM thereby defining its semantics. Furthermore, we use a definition of reachable states $\mathit{ReachableStates}_{BM}$ for a Bip model BM which is defined inductively in a way that the initial state is reachable and all succeeding states of a reachable state are reachable.

The task of verifying deadlock-freedom is performed by using the refinement shown in Figure 2:

1. The top line in the figure shows our notion of deadlock-freedom for a Bip model. We ultimately want to prove this line. We demand that all reachable states have at least one succeeding state. Thus, there is no reachable state where no transition is possible.

2. Instead of a direct proof, we follow the architecture of D-Finder and take advantage of the invariants discovered by D-Finder (Ψ_{BM},$\neg \mathit{DIS}_{BM}$): we conduct the proof shown in the second line consisting of two proof goals. The first goal reformulates the notion of enabled states and puts a predicate $\neg \mathit{DIS}_{BM}$ instead. Thus, we may verify that this goal

holds for a Bip model BM. The second proof goal [PO1] (Proof Obligation 1) states that whenever one proves the first goal correct, the correctness property of the first line is implied – thereby guaranteeing the more human readable notion of correctness.

3. The third line splits the first proof goal of the second line into two proof goals [PO2] and [PO3]. This line introduces an invariant $\Psi_{BM}(s)$ as a transitive step. This invariant is part of the certificate. To use this line in our proofs we have to show that it also implies the first line.

The $\neg DIS_{BM}$, and Ψ_{BM} are provided by D-Finder.

Most tasks in this proof scheme are relatively easy from a technical point of view. It is sufficient to prove the three proof obligations and construct the proof for the first line via transitivity of the implication rule. This, as well as proving [PO1] and [PO3] can easily be done automatically. However, the generation of the invariant $\Psi_{BM}(s)$ can be error prone. Therefore the automatic verification that

[PO2] $\forall s.\mathit{ReachableStates}_{BM}(s) \longrightarrow \Psi_{BM}(s)$

does hold is a challenging tasks of our methodology. It captures the correctness of the main task of the D-Finder tool: finding invariants. The work presented in the rest of this paper concentrates on generating realistic invariants, reports on experiences with case studies and suggests ways to improve the computation of the invariants thereby making the verification task easier.

1.2 Overview

We introduce the BIP semantics for modeling our systems in Section 2 and present a small example. A discussion of invariants of BIP models and their properties is given in Section 3. Section 4 introduces robust BIP models and a motivation for and proofs of their properties. The benefits of robust BIP models in verifying invariants for our example application scenario are presented in Section 5. Related work is discussed in Section 6. In Section 7 we draw a conclusion and present our goals for future work.

2 BIP Models and their Semantics

In this section we describe the semantics of Bip models. Bip is a software framework designed for building embedded systems consisting of heterogeneous components. It is characterized by three modeling layers: behavior of components encoded as transition systems extended with variables, interactions between components realized via communication ports and priority rules which reduce non-determinism between interactions (BIP stands for Behaviors + Interactions + Priorities). Apart from code generation the Bip tool chain comprises static analyses tools for checking properties like deadlock-freedom.

Bip models are composed of atomic components [BBS06, BBSN08] that can be composed into larger components. Components are state transition systems. They communicate via ports with each other.

Definition 2.1 (Atomic BIP component). *An atomic component B_i can be represented by a tuple*
(L_i, P_i, T_i, V_i) such that

- *V_i is a set of variables,*
- *$L_i = \{l_i^0, l_i^1, l_i^2, ..., l_i^k\}$ is a set of control locations,*
- *P_i is a set of ports,*
- *$T_i \subseteq L_i \times (X_i \to bool) \times (X_i \to X_i) \times P_i \times L_i$ is a set of transitions, each one comprising a location, a guard function $g : X_i \to bool$, an update function $f : X_i \to X_i$, a port, and a succeeding location. The X_i denote valuation functions: mappings from variables V_i to their values D_i.*

The guard functions are predicates and are formulated on the variables appearing in an atomic component. The following definition describes a language for these predicates which we use for the work presented in this paper:

Definition 2.2 (Guard language)**.** *A predicate ϕ belongs to the guard language iff it is constructed using the following rules:*

$$\begin{aligned}
&\phi ::= \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid e\\
&e ::= e' < e' \mid e' \leq e' \mid e' = e' \mid e' \neq e' \mid e' \geq e' \mid e' > e'\\
&e' := op \mid e' + e' \mid e' - e' \mid val \cdot e'\\
&op ::= var \mid val
\end{aligned}$$

Assuming the guard function appears in the ith component, the $var \in V_i$ are variables appearing in it. $val \in D_i$ denotes some numerical type. The variables $var \in V_i$ are mapped to the same type. Typical types are reals and integers.

The semantic interpretation of the guard language follows the rules of predicate logic and arithmetic. Note, that the expressibility of the guard language corresponds to Presburger arithmetic when D_i denotes integer values.
The atomic components of a BIP model are connected via ports. They communicate via interactions. Thus, a composed component is defined as a tuple $((B_1, ..., B_n), \mathit{Interactions})$ comprising the atomic components and their interactions.
An interaction is a tuple $(p_1, \ldots, p_n)$ where p_i is a port of the atomic component B_i or $\perp$ if B_i is not involved in this interaction.
The state of an atomic component B_i is a tuple (l_i, x_i) comprising a location and a variable valuation function. The state of a BIP model is the product of the state of its atomic components: $(L_1 \times X_1) \times \ldots \times (L_n \times X_n)$.
A transition relation for BIP models is defined via the following predicate.

Definition 2.3 (Transition Relation)**.** *A transition relation for a* BIP *model BM (denoted $[\![BM]\!]_{BIP}$) for* BIP *models is defined via the following rule:*

$$\frac{\begin{array}{c}(p_1, \ldots, p_n) \in \mathit{Interactions}\\ \forall i \in \{1..n\}.\ (l_i, g_i, f_i, p_i, l'_i) \in B_i \wedge (g_i(x_i) \wedge x'_i = f_i(x_i)) \vee (l_i = l'_i \wedge p_i = \perp \wedge x'_i = x_i)\end{array}}{(((l_1, x_1), ..., (l_n, x_n))\ ,\ ((l'_1, x'_1), ..., (l'_n, x'_n))) \in [\![BM]\!]_{BIP}}$$

A state transition from a given reachable state is possible if there is an interaction such that there is in each component either a possible state transition labeled with the port or the component is not involved in the interaction. Furthermore, in order to do a transition of an atomic component the appropriate guard functions must evaluate to true. To derive the

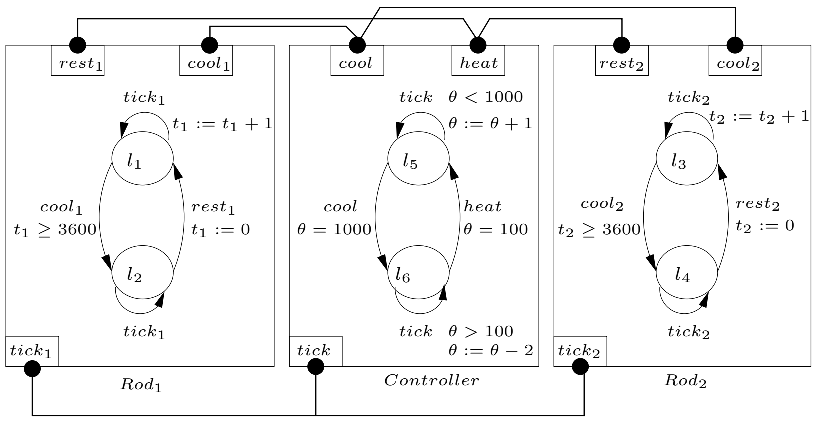

Figure 3: Temperature Control System

succeeding states the update functions are performed on the valuation functions of the involved atomic components.

Using the transition relation, reachable states of a Bip model are defined in the following definition.

Definition 2.4 (Reachable States). *The predicate ReachableStates$_{BM}$ indicating reachable states of a* Bip *model BM with an initial state* $s_0 \in (L_1 \times X_1) \times \ldots \times (L_n \times X_n)$ *is defined via the following inductive rules:*

$$\frac{}{\mathit{ReachableStates}_{BM}(s_0)}$$

$$\frac{\mathit{ReachableStates}_{BM}(s) \quad (s, s') \in [\![BM]\!]_{BIP}}{\in \mathit{ReachableStates}_{BM}(s')}$$

The first rule says that the initial state is reachable. The second inference rule captures the transition behavior of Bip using the transition relation.

An Example Figure 3 shows a temperature control system [BBSN08, ACH+95] modeled in Bip. It controls the cooling of a reactor by moving two independent control rods. It is a simple example to illustrate D-Finder and its invariants. The goal is to keep the temperature between $\theta = 100$ and $\theta = 1000$. When the temperature reaches the maximum value one of the rods has to be used for cooling. The Bip model comprises three atomic components one for each rod and one for the controller. Each contains a state transition system. Transitions can be labeled with guard conditions, valuation function updates, and a port. The components interact via ports thereby realizing cooling, heating, and time elapsing interactions. Initially the system starts in locations l_1, l_3, l_5 with values of $t_1 = 3600$, $t_2 = 3600$, $\theta = 100$. Note, that the system does indeed contain a deadlock. Since we are interested in verifying that invariants hold, this, however, is not important in the context of this paper.

3 Invariants of BIP Models

In this section we discuss invariants for Bip models and motivate desired properties.

Definition 3.1 (Invariant)**.** *A predicate I over the states of a* Bip *model BM is an invariant of BM iff*
$\forall s.ReachableStates_{BM}(s) \longrightarrow I(s)$

The generated invariants Ψ_{BM} that we are verifying in our theorem prover are composed of component invariants (CI) and interaction invariants (II):

$$\Psi_{BM} \overset{def}{=} CI_1 \wedge ... \wedge CI_n \wedge II_1 \wedge ... \wedge II_m$$

The following invariants are computed by D-Finder to approximate the behavior of the components (*component invariants*) in the example from Figure 3:

- $CI_1 = (at_{l1} \wedge t_1 \geq 0) \vee (at_{l2} \wedge t_1 \geq 3600)$
- $CI_2 = (at_{l3} \wedge t_2 \geq 0) \vee (at_{l4} \wedge t_2 \geq 3600)$
- $CI_3 = (at_{l5} \wedge 100 \leq \theta \leq 1000) \vee (at_{l6} \wedge 100 \leq \theta \leq 1000)$

at_i is a predicate denoting the fact that we are at location i in a component. In addition to component invariants D-Finder computes interaction invariants capturing the behavior induced by interactions between the atomic components. In addition to component invariants D-Finder generates interaction invariants which capture the behavior of components interacting with each other. An example for an interaction invariant for the given Bip model is shown below:

$II_1 = (at_{l1} \wedge t_1 = 0) \vee (at_{l3} \wedge t_2 = 0) \vee (at_{l5} \wedge 101 \leq \theta \leq 1000) \vee (at_{l6} \wedge (\theta = 1000 \vee 100 \leq \theta \leq 998))$

Definition 3.2 (Inductive Invariants)**.** *An invariant I is inductive for a* Bip *model BM iff*

1. *It holds for the initial state s_0 of BM: $I(s_0)$*
2. $\forall s\ s'.I(s) \wedge (s, s') \in [\![BM]\!]_{BIP} \longrightarrow I(s')$

By using the definition of reachable states we can prove the following theorem which is independent of concrete Bip models:

Theorem 3.1. *Every inductive invariant of a* Bip *model BM is also an invariant of BM.*

Both CI_1 and CI_2 are inductive. CI_3 is not inductive because it evaluates to true for a state where we are at control location $l6$ (at_{l6}) and $\theta = 101$. But, it does not hold for the succeeding state at_{l6} and $\theta = 99$. Note, that since a state where at_{l6} and $\theta = 101$ is never reached within a real system run, CI_3 is still an invariant.

For verifying invariants during certificate checking we perform an induction on the set of reachable states. For this reason, it is highly desirable if invariants are inductive.

Making Invariants Inductive by Strengthening We can make invariants inductive by strengthening them. Given an invariant I we can add some strengthening constraints C to create a stronger invariant I'. The proof that I' implies that I holds for a given state s is a trivial application of the conjunction elimination rule:

$$I'(s) \overset{def}{=} I(s) \wedge C(s) \longrightarrow I(s)$$

For example CI_3 can be made inductive by adding a constraint that at location $l6$, θ is always divisible by two $(2|\theta)$. We can now verify the inductive invariant and show that it implies the weaker non-inductive one. Thus, the non-inductive one, is proven to be an invariant, too.

We have experimented with techniques to strengthen invariants automatically. In many cases it is possible to "guess" the C constraints that make invariants inductive. The additional constraint could be constructed following the general method presented in [BM08] that refines the invariant to reach an inductive one.

However, sometimes strengthening invariants seems artificial. Adding the divisibility constraint mentioned above to a variable that represents a physical measure could also indicate some design flaw of the original system. We should not base the verification of a system on the fact that the temperature measured by some sensors is an even number.

So, instead of strengthening the constraints on physical measures, we introduce in this paper a way to model the uncertainty of measurement. We concentrate on finding slightly weaker invariants of systems that represent the nature of variables depending on physical measurements in a more natural way.

For each invariant there is always a weaker invariant that is inductive: $I \equiv \mathit{true}$ is the weakest invariant for all Bip models and is inductive.

4 Robust BIP Models

In this section we introduce robust Bip models. Robust Bip models and their invariants are aimed at describing systems in a more natural way. Specifically, the target of our approach are systems whose values represent physical measurements. These are performed by sensors, which are usually not exact but have some tolerance range of imprecision associated with them. The measured value can vary within this range differing from the actual value. To capture the behavior in Bip models that are based on this imprecise information, we introduce special sets of *measurement* variables. Guard functions depending on an exact *measurement* variable in a Bip model are changed in a way that they evaluate to true – i.e. may perform a transition – within a range of unpreciseness to achieve robust Bip models.

Robust Bip models are realized by exchanging guards by robust guards described by a robust guard language:

Definition 4.1 (Robust guard language). *A robust guard language describing predicates ϕ is defined for a set of measurement variables $V_R \subseteq V_i$ if the guard appears in the ith component in the following way:*

$$\begin{aligned}
\phi &::= \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid e \\
e &::= e' < e' \mid e' \leq e' \mid e' = e' \mid e' \neq e' \mid e' \geq e' \mid e' > e' \\
e' &::= op \mid e' + e' \mid e' - e' \mid val \cdot e' \\
op &::= var \mid val \mid m + \delta_m
\end{aligned}$$

$$
\begin{array}{ll}
\langle(\phi_1 \wedge \phi_2)\rangle_2 = \langle\phi_1\rangle_2 \wedge \langle\phi_2\rangle_2 & \langle(\phi_1 \vee \phi_2)\rangle_2 = \langle\phi_1\rangle_2 \vee \langle\phi_2\rangle_2 \\
\langle(e'_1 < e'_2)\rangle_2 = \langle e'_1\rangle_2 < \langle e'_2\rangle_2 & \langle(e'_1 \leq e'_2)\rangle_2 = \langle e'_1\rangle_2 \leq \langle e'_2\rangle_2 \\
\langle(e'_1 \geq e'_2)\rangle_2 = \langle e'_1\rangle_2 \geq \langle e'_2\rangle_2 & \langle(e'_1 > e'_2)\rangle_2 = \langle e'_1\rangle_2 > \langle e'_2\rangle_2 \\
\langle(e'_1 = e'_2)\rangle_2 = \langle e'_1\rangle_2 = \langle e'_2\rangle_2 & \langle(e'_1 \neq e'_2)\rangle_2 = \langle e'_1\rangle_2 \neq \langle e'_2\rangle_2 \\
\langle(e'_1 + e'_2)\rangle_2 = \langle e'_1\rangle_2 + \langle e'_2\rangle_2 & \langle(e'_1 - e'_2)\rangle_2 = \langle e'_1\rangle_2 - \langle e'_2\rangle_2 \\
\langle(val \cdot e')\rangle_2 = val \cdot \langle e'\rangle_2 & \langle val\rangle_2 = val
\end{array}
$$

$$
\begin{array}{ll}
\langle m\rangle_2 = m + \delta_m & \text{if } m \in V_R \\
\langle var\rangle_2 = var & \text{otherwise}
\end{array}
$$

Figure 4: Function for Introducing the δ_m

$var \in (V_i \backslash V_R)$, $m \in V_R$, val denotes some numerical type var like reals or integers.

Each reference to a measurement variable within a robust guard function is accompanied by some unknown constant δ_m which captures its imprecision. Compared to the original guard language, the interpretation of the δ_m in the robust guard language requires some non-trivial definitions.

The semantical interpretation of guard functions is adapted in a way that a robust system is an abstraction of a non-robust system. This means that due to non-determinism it allows more possibilities of transition but preserves the transition possibilities of the non-robust system. Given a set of measurement variables V_R, for each $m \in V_R$ there is some unknown δ_m within some fixed range $-\Delta_m \leq \delta_m \leq \Delta_m$ ($\Delta_m \geq 0$). This range captures the level of imprecision which a value that represents a physical measurement can have.

A robust guard is constructed from a non-robust guard in two steps:

- In the first step, negations are eliminated by putting them to the lowest level thereby changing (in)equalities using a function $\langle ... \rangle_1$. This function is defined inductively on the term structure of the guards and performs e.g. the following transformations:
  - $\langle\neg(\phi_1 \wedge \phi_2)\rangle_1 = \langle\neg\phi_1\rangle_1 \vee \langle\neg\phi_2\rangle_1$
  - $\langle\neg(e'_1 > e'_2)\rangle_1 = (e'_1 \leq e'_2)$
  - $\langle\neg(e'_1 = e'_2)\rangle_1 = (e'_1 \neq e'_2)$

  Thus, it eliminates all cases of $\neg\phi$.
- The second step introduces the δ_m for the measurement variables and is performed by a function $\langle ... \rangle_2$ which is shown in Figure 4.

Consider as an example the guard $\neg\theta = 1000$. It is transformed into $\langle\langle\neg\theta = 1000\rangle_1\rangle_2 = \langle\theta \neq 1000\rangle_2 = \theta + \delta_\theta \neq 1000$ for a measurement variable θ.
The semantic interpretation of such a guard ϕ is done in the following way for measurement variables $m_1 ... m_n$:

$$\exists\delta_{m_1} ... \delta_{m_n}.\phi \text{ with } -\Delta_{m_1} \leq \delta_{m_1} \leq \Delta_{m_1} \ ... \ -\Delta_{m_n} \leq \delta_{m_n} \leq \Delta_{m_n}.$$

Thus, in the above example, we have $\exists\delta_\theta.\theta + \delta_\theta \neq 1000$ with $-\Delta_\theta \leq \delta_\theta \leq \Delta_\theta$.

The existential quantification of the δ_m ensures that the robust guard function over approximates the corresponding non-robust guard function. We need the first transformation step eliminating the negations, because our methodology only works, if there are no negations in our guard: an existential quantification over a negated δ_m would result in an under approximation.

Note, that robust guards can be transformed back into the non-robust guard language while preserving their semantics. $\theta + \delta_\theta \neq 1000$ can be equivalently written as $1000 - \Delta_\theta > \theta \vee 1000 + \Delta_\theta < \theta$ by only using the constant Δ_θ. For practical applications, the computations performed in D-Finder can be done, using either the robust guards having been translated back into the non-robust guard language, or with slight modifications in D-Finder, by using the robust guard language directly.

A non-robust guard function can be constructed from a robust guard function by setting each δ_m to zero.

Definition 4.2 (Robust BIP Models)**.** *A BIP model is considered robust for a set of variables V_R iff all guard functions depending on a variable $m \in V_R$ are formalized in the robust guard language.*

Each guard can be substituted by a robust guard by replacing each occurrence of $m \in V_R$ by $m + \delta_m$ as shown above. We define a function $Robust_{V_R}$ for a set of measurement variables V_R to map BIP models to robust BIP models by replacing the guard functions by robust ones.

Theorem 4.1 (Reachable State Inclusion)**.** *For a robust BIP model BM_Δ and a BIP model BM with $BM_\Delta = Robust_{V_R}(BM)$, given a set of measurement variables V_R the following property holds :*
$ReachableStates_{BM} \subseteq ReachableStates_{BM_\Delta}$

Proof:
We have to show that the robust guard functions allow at least all state transitions that the non-robust guard functions allow. We do an induction on the term structure of ϕ to show:

$\langle\langle ... \rangle_1\rangle_2$ evaluates at least in all cases to true where $\delta_{m_1}...\delta_{m_n}$ fixed to 0 would evaluate to true.

An example robust BIP model constructed from our temperature controller example (cp. Figure 3) for a measurement variable θ is shown in Figure 5.

Theorem 4.2 (Invariant Preservation)**.** *Each Invariant I of a robust BIP model BM_Δ is an invariant of a BIP model BM for a given set of measurement variables V_R if $BM_\Delta = Robust_{V_R}(BM)$*

Proof:

$\forall s \in ReachableStates_{BM_\Delta}.I(s)$ (since I is an invariant of BM_Δ) and
$ReachableStates_{BM} \subseteq ReachableStates_{BM_\Delta}$ (reachable states inclusion) implies
$\forall s \in ReachableStates_{BM}.I(s)$ □

5 Application of Robust BIP Models

In this section we discuss the deadlock checking of robust BIP models with D-Finder and summarize and extend our comparison of invariants used in the process of certifying the deadlock freedom of a BIP model for usage with a higher-order theorem prover after D-Finder has provided its verdict.

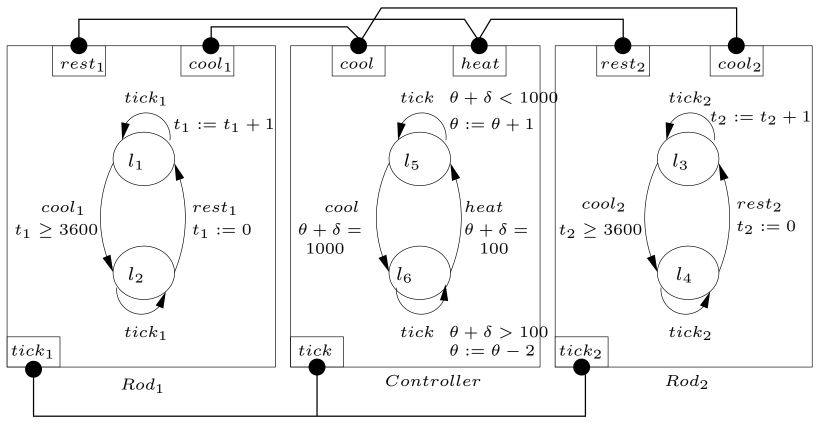

Figure 5: Robust Temperature Control System with Measurement Variable θ

$$\begin{tikzcd}
 & Enabled_{BM} \quad (deadlock\text{-}freedom) & \\
\text{(D-Finder) } \Psi_{BM_\Delta} \arrow[ur, Rightarrow, "SMT\ solving"] & \text{(D-Finder) } \Psi_{BM} \arrow[u, Rightarrow, "SMT\ solving"] & \\
 & & \Psi_{BM} \wedge Guess \arrow[ul, Rightarrow, "trivial"'] \\
ReachableStates_{BM_\Delta} \arrow[uu, Rightarrow, "ind.\ proof(new\ proof\ strategy)"'] & & \\
 & & ReachableStates_{BM} \arrow[ull, Rightarrow, "trivial", "BM \equiv (BM_\Delta \wedge \Delta{=}0)"'] \arrow[uu, Rightarrow, "ind.\ proof"'] \arrow[uuul, Rightarrow, dashed, "original\ proof\ strategy"']
\end{tikzcd}$$

weaker & likely inductive **non-inductive invariant** **stronger & inductive**

Figure 6: Relation Between Invariants

Invariants of Robust and Non-robust BIP models and D-Finder The relations between different invariants of BIP models BM and its robust counter-part BM_Δ is shown in Figure 6

Starting from the weakest inductive invariant (*true*), D-Finder computes a sequence of stronger invariants using the initial conditions and the provided BIP model BM. Due to abstractions performed by D-Finder this process can provide an invariant Ψ_{BM} that is not inductive. Using non-robust BIP models in our certification process the proof of the (dashed) implication can not be established for non-inductive invariants and the certificate generation fails. The critical abstraction used in D-Finder is in elimination of existential quantifiers in the logical formula that defines the successors of a state which becomes part of the provided invariant. D-Finder uses the Omega library for that step [Ome00]. A closer look at the original formula and its abstraction reveals the constraints that were lost during abstraction. The loss of precision is due to the fact that (1) some facts are implicit (*e.g.*, for a variable x, the fact $x \geq 0$ is eliminated if x is of type `nat`), or (2) facts cannot be represented in the logic (this is the case for divisibility constraints in the Omega library). The addition of these lost constraints leads to an inductive invariant. The divisibility constraints can be useful for variables of the program that range over discrete domains (*e.g.*, counters). However it produces inductive but unrealistic invariants for variables that represent physical measurements provided by sensors.

Furthermore, Figure 6 shows the goals sketched in the introduction and in Section 3 on invariants. In order to conduct the proof we seek for an inductive invariant $\Psi_{Inductive}$ that contains the (non-computable) sets of reachable states and that entails the deadlock-freedom property ($Enabled_{BM}$), such that the following implications holds:

$$ReachableStates_{BM} \Rightarrow \Psi_{Inductive} \qquad\qquad \Psi_{Inductive} \Rightarrow Enabled_{BM}$$

To build $\Psi_{Inductive}$ we can try to strengthen the invariant provided by D-Finder by guessing a suitable constraint such that $\Psi_{BM} \wedge Guess$ is inductive. The certificate then encapsulates the proof of the right-most chain of implications. Otherwise we can try the approach promoted in this paper for building an inductive invariant weaker than Ψ_{BM} but still strong enough to entail deadlock-freedom. This approach comprises the Ψ_{BM_Δ} invariant of the robust BIP model and corresponds to the left-most chain of implications of Figure 6.

Each approach can lead to a certificate based on a proof by induction which can be automatically generated and then provided to a higher-order theorem prover for checking.

Evaluation In contrast to non-robust BIP models, D-Finder produces inductive invariants of robust BIP models in the case studies that we examined so far. The model of Figure 5 is the robust version of our running example of Figure 3. In all guards the uncertainty δ is added to the θ variable that corresponds to the measure of the temperature. This transformation prevents the generation of over constrained invariants. Obviously, the generated invariants are less precise than those obtained for the original model. Hopefully, in our experimentations this did not introduce new deadlock possibilities. Actually, it is highly desirable that the deadlock-freedom property of a system does not depend on the sensitivity of sensors.

Figure 7 summarizes our approach using the running example. It shows the invariant CI_3 generated by D-Finder on the original BIP model BM (in the middle) compared to the corresponding stronger invariant obtained by strengthening (on the left) and the weaker but robust invariant generated by D-Finder from the robust model BM_Δ (on the right).

The CI_3 invariant relates the value of θ to the location of the controller state machine. The original invariant is not inductive. It can be either strengthened by adding the additional constraint, $2|\theta$. This leads to an inductive but unrealistic invariant. A more realistic invariant

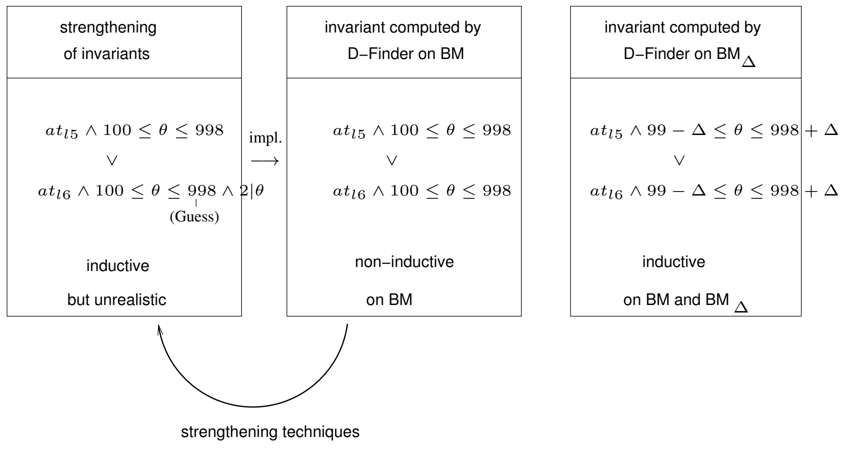
strengthening of invariants
$at_{l5} \wedge 100 \leq \theta \leq 998$
$\vee$
$at_{l6} \wedge 100 \leq \theta \leq 998 \wedge 2|\theta$
(Guess)
inductive but unrealistic
impl.
invariant computed by D−Finder on BM
$at_{l5} \wedge 100 \leq \theta \leq 998$
$\vee$
$at_{l6} \wedge 100 \leq \theta \leq 998$
non−inductive on BM
invariant computed by D−Finder on BM$_\Delta$
$at_{l5} \wedge 99 - \Delta \leq \theta \leq 998 + \Delta$
$\vee$
$at_{l6} \wedge 99 - \Delta \leq \theta \leq 998 + \Delta$
inductive on BM and BM$_\Delta$
strengthening techniques

Figure 7: Handling the Example Invariant

is obtained by running D-Finder on the robust model. The resulting invariant is inductive and strong enough to conduct the last step of the deadlock analysis. The final result is as precise as the ones obtained with the two other invariants; no false deadlocks are generated by robust invariants.

Not all invariants of robust BIP models have to be necessarily inductive. It may be possible that we need to strengthen them sometimes e.g. by using the technique discussed in [BM08]. These techniques, however, must not add constraints bearing unnatural facts on the measuring process of the measurement variables.

6 Related Work

This paper describes the modification of systems in order to generate realistic inductive invariants for a verification tool. These invariants are verified to hold by a higher-order theorem prover for certifying the results of the verification tool. Inductiveness is crucial for deductive proofs in the certification process. To our knowledge this particular subject has not yet been studied before.

Certification The certification of analysis results is an important aspect of this work. We generate certificates in the form of proof-scripts for a higher-order theorem prover. The generation of proofs to certify the verdict of a model-checker was first introduced in [Nam01]. Other important work for certifying the results of verification tools comprise the use of support sets for a model checker [TC02], keeping track of justifications for the BLAST model-checker [HJM+02], and a SAT solver that generates certificates [ZM03]. Further related is Proof-Carrying Code (PCC) [Nec97], a method to guarantee that executable code fulfills a policy on access and resource management and Foundational PCC [WAS03] characterized by a small set of axioms and a simpler proof-checker. In this paper we concentrate on certifying invariants via a formal proof done by induction.

Generation of Inductive Invariants Automatic generation of invariants has been studied for a long time (see e.g. [MP95]). For the invariants considered in this paper the papers [BLS96] and [SDB96] where most influential. [BLS96] describes many principles that have been implemented in our deadlock-verification tool D-Finder.

[BM08] focuses on techniques to incrementally generate inductive invariants. The main idea is to use counter-examples to refine an invariant until it becomes provable by induction. The paper features in addition to the description of general techniques, further valuable techniques to refine invariants by splitting large conjunctions into subparts and refine selected subparts independently. This method has been successfully used in the refinement of boolean invariants. The presented techniques require a deep knowledge of the class of systems in consideration, in order to cleverly take advantage of the presented counter-example guided refinement approach of invariants.

In this paper we regard strengthening of non-inductive invariants by taking advantage of our knowledge of the verification tool D-Finder – we know when the invariant can lose their inductiveness quality – as one possibility to achieve inductive invariants.

Robustness Another feature of this paper is robustness. Robustness of timed automata is described in [GHJ97]. This notion is similar to our notion and has been introduced for real-time systems in order to cope with properties that e.g. occur when transforming continuous signals to discrete values or problems due to imprecision of sensors. The theory of verification of systems is enriched with a so called tube-semantics to capture uncertainty. Similarly, [AM95] introduces the notion of finite variability for properties of continuous systems to capture semantics intervals of continuous time. The presented method allows proving properties of a continuous semantics by reasoning on a discrete semantics which is more appropriate for automated verification and deductive reasoning. A constrained solving based method for generating inductive invariants for hybrid systems is presented in [SSM04].

In contrast to these work, we do not consider real-time system with continuous semantics. The BIP system has a discrete semantics and acquires information about the physical quantities through sensors. One of our our goals is introducing uncertainty about the sensors measurements while reusing D-Finder invariant generation techniques based on a discrete semantics.

7 Conclusion and Future Work

In this paper we have introduced the notion of robust BIP models for systems containing values representing the results of physical measurements. Robust BIP models can be obtained from non-robust BIP models and over approximate their behavior. We proved that each invariant of the robust BIP model is also an invariant of the corresponding non-robust BIP model. We motivated that invariants that have been automatically generated by the deadlock detection tool D-Finder for robust BIP models tend to be inductive while those of non-robust systems are likely to be non-inductive. Inductivity of invariants allows for easy formal verification that they do indeed hold in a higher-order theorem prover. Our technique is developed for certifying the results of D-Finder at runtime. It allows to keep a discrete representation instead of a continuous representation of values occurring in a system, thus enabling the use of tools like D-Finder which work on discrete representations.

Future work continuing this on-going work involves the analysis of further case studies to discover more benefits and potential limitations of our approach. We have achieved first results in producing realistic and inductive invariants by running D-Finder on robust BIP models. It seems that instead of using strengthening techniques, we can use robust (and therefore weaker) constraints to achieve inductive invariants. Once D-Finder provides analyzes for BIP models with value exchange during component interaction, we will model sensors and provided values as distinct components. Future work includes further reasoning on robustness to ensure that inductiveness of D-Finder invariants is guaranteed by the construction process of robust BIP models. More technical challenges comprise the addition of ranges of unpreciseness that can deal with non-linear errors such as a certain error percentage of a measured value. This can be modeled as functions depending on measured values and have to be integrated in our guard language. These functions are difficult to handle since the Yices SMT-solver can only deal with invariants generated from guard functions formalized in Presburger arithmetic. Another task reserved to future work is fixing invariants by adding constraints in order to make them inductive. Such constraints might be generated by using hints from unresolved theorem prover goals created during the process of trying to prove an invariant correct.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[AM95] L. De Alfaro, Z. Manna. Verification in Continuous Time by Discrete Reasoning. Proceedings of the 4-th International Conference on Algebraic Methodology and Software Technology AMAST'95, LNCS, Springer-Verlag, 1995.

[BBSN08] S. Bensalem, M. Bozga, J. Sifakis, and T-H. Nguyen. Compositional Verification for Component-based Systems and Application. ATVA 2008 6th International Symposium on Automated Technology for Verification and Analysis, October 20-23, 2008, Seoul, South Korea.

[BBS06] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *SEFM*, pages 3–12, 2006.

[BLS96] S. Bensalem, Y. Lakhnech, H. Saidi. Powerful techniques for the automatic generation of invariants. CAV'96, Springer-Verlag, 1996.

[BM08] A. R. Bradley and Z. Manna. Property-directed incremental invariant generation. Formal Aspects of Computing, Springer-Verlag 2008.

[BP08] J. O. Blech and M. Périn. Towards certifying deadlock-freedom for BIP models. Technical Report TR-2008-1, Verimag, September 2008.

[BP08a] J. O. Blech and M. Périn. On Certificate Generation and Checking for Deadlock-freedom of BIP Models. Technical Report TR-2008-19, Verimag, December 2008.

[DM06] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). CAV'06. Volume 4144 of LNCS, Springer-Verlag 2006.

[GHJ97] V. Gupta, T. A. Henzinger, R. Jagadeesan. Robust Timed Automata Proceedings of the International Workshop on Hybrid and Real-Time Systems, LNCS, Springer-Verlag, 1997.

[HJM+02] T. A. Henzinger, R. Jhala, R. Majumdar, G. C. Necula, G. Sutre, and W. Weimer. Temporal-safety proofs for systems code. *Proc of CAV '02*, 2002. Springer-Verlag.

[Nam01] K. S. Namjoshi. Certifying model checkers. *Proc of CAV '01*, 2001. Springer-Verlag.

[Nec97] G. C. Necula. Proof-carrying code. In *POPL '97*, pages 106–119, New York, NY, USA, 1997. ACM.

[MP95] Z. Manna and A. Pnueli. Temporal Verification of Reactive Systems: Safety. Springer-Verlag, 1995.

[Ome00] The Omega library. Version 1.2 (August 2000)

[SDB96] J. X. Su, D. L. Dill, C. W. Barrett. Automatic Generation of Invariants in Processor Verification. FMCAD'96, 1996.

[SSM04] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing Invariants for Hybrid Systems. In Hybrid Systems: Computation and Control (HSCC 2004), LNCS, Springer-Verlag 2004.

[TC02] L. Tan and R. Cleaveland. Evidence-based model checking. *Proc of CAV '02*, London, UK, 2002. Springer-Verlag.

[WAS03] D. Wu, A.W. Appel, and A. Stump. Foundational proof checkers with small witnesses. *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming*, 2003.

[ZM03] L. Zhang and S. Malik. Validating SAT Solvers Using Independent Resolution-Based Checker: Practical Implementations and Other Applications DATE 2003.