PyEcdar: Towards Open Source Implementation for Timed Systems
Axel Legay, Louis-Marie Traonouez

To cite this version:

HAL Id: hal-01087318
https://hal.archives-ouvertes.fr/hal-01087318
Submitted on 25 Nov 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
**PYEDAR: Towards Open Source Implementation for Timed Systems**

Axel Legay, Louis-Marie Traonouez

INRIA Rennes, France, firstname.lastname@irisa.fr

**Abstract.** PYEDAR is an open source implementation for reasoning on timed systems. PYEDAR’s main objective is not efficiency, but rather flexibility to test and implement new results on timed systems.

1 **Context**

To solve complex problems such as scheduling tasks in embedded applications, the ability to reason on real time is mandatory. It is thus not a surprise that, over the last twenty years, the rigorous design of real-time systems has become a main research topic. Among major successes in the area, one finds the UPPAAL toolset [1] that is promoted by industries, and that has been used to verify complex properties of complex protocols such as the Herschel-Planck, the root contention protocol, or Audio-Control Protocol developed by Philips. Recently, timed tools have been extended to reason not only on the properties of the system, but also on the effects of its interactions with a potentially unknown environment. Tools such as UPPAAL-TIGA do this via game-theory [2]. The code of UPPAAL and related toolsets is not available and their interfaces are fixed in stone. Those choices shall not been seen as drawbacks, but rather as strategic choices for an industrial dissemination. However, from a scientific point of view, this makes it hard for researchers to reuse part of those toolsets to quickly implement and evaluate their new results without sharing them with tool makers.

We present PYEDAR (https://project.inria.fr/pyedar/) that is a new python implementation of well-known results on timed systems and games. We then show that the tool can be used to implement new results in timed systems. Our main objective with PYEDAR is to offer an open source platform to quickly test new results on timed systems. Of course, this implementation is not as competitive as well-established toolsets, but it is very flexibility and easy to use and extend.

2 **The PYEDAR Toolset in a Nutshell**

As a foundation to develop new algorithms, PYEDAR offers an implementation of the reachability analysis for timed automata as well as an implementation of the forward algorithm from [2] that is used to solve reachability problem for timed games. Then, the tool offers the implementation of a series of brand new results on timed systems. The first is the timed specification theory from [3] that has been developed to reason on complex systems described as a combination of components. The specifications of those components are given by Timed Input/Output Automata (TIOA), where inputs represent behaviors of the environment, and outputs those of the system. The tool is...
able to 1. decide whether an implementation (e.g. Fig. 1b) conforms to a given specification (e.g. Fig. 1a), decide whether a specification can be implemented (consistency), 3. compare specifications (refinement – timed game), 4. logically/structurally compose two specifications, 5. synthesize a specification from a set of requirements (quotient), and 6. prune states from which the environment has no strategy to avoid bad behaviors (compatibility) – the operation requires the implementation of a timed game. The theory has also been implemented in ECDAR [4]. An advantage of PYEDCAR is that its internal data structures can be used to save (and reuse) the result of composing/synthesizing specifications, while the one of ECDAR cannot. Also, PYEDCAR can perform compatibility on combined systems while ECDAR cannot.

PYEDCAR also offers the implementation of an extension of [5] to decide whether an implementation automaton is robust: i.e. if it remains conform to a specification when its output guards (resp. input) are exceeded (resp. restricted) by some $\Delta$ value. The tool can also synthesize the maximal $\Delta$ for which the implementation remains robust. The results extend to all the operations of the theory [6]. As an example, implementation of Fig. 1b is robust with respect to specification of Fig. 1a up to $\Delta = 1$. Beyond that point, the perturbations of the output transitions, which is $5 - \Delta \leq y \leq 5 + \Delta$, exceeds the guard $y \geq 4$ and the invariant $y \leq 6$ of the specification. It is worth mentioning that the internal structure of ECDAR does not permit to implement robustness on top of the specification theory. So, albeit the work in [6] is an extension of the one in [3], using ECDAR would require an entirely new implementation. Several robustness theories for timed automata have been implemented in tools such as shrinktech [7], but PYEDCAR is the first to offer this feature for a complete specification theory.

Finally, PYEDCAR offers the ability to reason on variability [8]. There, the model is an extended timed automata that permits to represent features of both the system and its environment – such features may appear or disappear at runtime. As an example, Fig. 1c represents a specification of the system using two adaptive features for the environment (out_of_tea and out_of_cof). These features may be enabled or disabled at runtime during input transitions, which may restrict the possible behaviors of the system. PYEDCAR exploits an extension of timed game algorithms to synthesize e.g., the minimal set of features that are needed by a system so that it verifies a timed CTL property, whatever the environment does. To the best of our knowledge, PYEDCAR is the first to offer a timed implementation of such a complex problem in software engineering.
3 Architecture of the Tool

**PYECDA** works inside an interactive python shell, and offers a set of modules and functions to load models and perform computations. In **PYECDA**, models are written by using the interface of UPPAAL or ECDAR, and uploaded via an XML file. Once the models have been loaded, the user can perform one or several queries via the shell in an on-demand manner.

**Input Language.** **PYECDA** supports the main language elements from ECDAR. That allows to design TIOAs with the syntax from [3], and additionally to use extended syntax elements, like constants and integer variables. See [https://project.inria.fr/pyecdar/](https://project.inria.fr/pyecdar/) for the grammar. TIOAs are specified with the ECDAR interface that is freely available, and then saved in XML. In case of features, Boolean variables are added to the model to witness the presence or absence of each feature. For the internal representation, **PYECDA** relies on the UPPAAL DBMs library used to represent the timing constraints of the model and a classical graph-based structure to represent its syntax. Contrary to ECDAR, **PYECDA** creates a dedicated structure for each component, including those that are obtained by combining existing ones. ECDAR is rigid and can only represent a new component by a pointer on states of the structures of those that participated to its creation. As a consequence, ECDAR cannot perform composability that consists in removing “bad states”. Indeed, since new components do not have their own structure, this operation would eventually remove states of individual components that participated to its creation and hence falsify the design. If features are present, then **PYECDA** combines BDDs used to logically represent sets of features on transitions with DBMs (see [8] for details). Finally, **PYECDA** also uses polyhedra, with bindings to the Parma Polyhedra Library, to encode parametric constraints in case the user wants to solve a robustness problem.

**Queries.** **PYECDA** offers two types of queries. The first one comes as a set of operators such as composition or quotient to build complex systems from small ones. The second type concerns operational queries such as the one of checking consistency, refinement, robustness, or properties of adaptive systems (see [https://project.inria.fr/pyecdar/](https://project.inria.fr/pyecdar/) for the complete list of queries). Depending on the problem to be solved, **PYECDA** outputs different kinds of results. As an example, if the tool is used to synthesize the set of features that allows to satisfy some temporal formula, this set is output as a binary expression. The tool can also be used to determine the winning states for a timed game, which allows to determine if the consistency, compatibility or refinement problems have been solved. Finally, using a counter-example refinement approach (CEGAR), it can compute the maximum perturbation allowed by the system to solve a robustness problem. **PYECDA** offers some extra features such as saving TIOAs into a new XML file so that they can be reused in other designs.

**Algorithms.** 1. **PYECDA** implements the on-the-fly safety game algorithm from [2] that is used e.g. to check consistency and refinement. The tool also uses a model transformation to reduce robust consistency/compatibility to consistency/compatibility and hence reuse the former algorithm. 2. The CEGAR algorithm is a parametric extension of the first [9] that allows to compute the maximal delta for which an implementation
remains robust. The last algorithms are backward propagation game algorithms [8] that compute the set of features that satisfies a formula for an adaptive system.

4 PYECDAR in Action

We quickly demonstrate how to use PYECDAR. Assume that the models presented in Fig. 1 are saved in an XML file machine.xml. We first load the XML file of the first two TIOAs:

\[
\begin{align*}
\text{In [1]: } & W = \text{pyecdar.loadModel("machine.xml")} \\
\text{In [2]: } & MS = W.getSpecification("MachineSpec"). \\
\text{In [3]: } & MI = W.getSpecification("MachineImpl").
\end{align*}
\]

We check if the implementation of Fig. 1b satisfies the specification in Fig. 1a:

\[
\begin{align*}
\text{In [4]: } & MI \leq MS \\
\text{Out[4]: } & \text{True}
\end{align*}
\]

We can then compute the maximum perturbation allowed by the implementation. This applies the CEGAR approach, starting with value 5, and with a confidence 0.1 for the result. The result is computed after 2 iterations and \(\Delta = 1\) is returned.

\[
\begin{align*}
\text{In [5]: } & \text{MI.maxRobSat(MS,5,0.1)} \\
\text{INFO:CEGAR: } & \text{New game with value 5} \\
\text{INFO:REACH: } & \text{2 states visited.} \\
\text{INFO:CEGAR: } & \text{...game is lost; refining...} \\
\text{INFO:CEGAR: } & \text{...refinement result: max=1 min=0 strict: False} \\
\text{INFO:CEGAR: } & \text{New game with value 1} \\
\text{INFO:REACH: } & \text{6 states visited.} \\
\text{INFO:CEGAR: } & \text{...game is won;} \\
\text{INFO:CEGAR: } & \text{...refinement result: max=1 min=1 strict: False} \\
\text{Out[5]: } & 1.0
\end{align*}
\]

Other examples, e.g. checking a temporal formula on the adaptive specification of Fig. 1c, are described on [https://project.inria.fr/pyecdar/](https://project.inria.fr/pyecdar/)

References