7-2013

PSyHCoS: Parameter synthesis for hierarchical concurrent real-time systems

Étienne ANDRÉ

Yang LIU

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Jin Song DONG

Shang-Wei LIN

PSyHCoS: Parameter Synthesis for Hierarchical Concurrent Real-Time Systems

Étienne André[1], Yang Liu[2], Jun Sun[3], Jin Song Dong[4], and Shang-Wei Lin[5,⋆]

[1] Université Paris 13, Sorbonne Paris Cité, LIPN, F-93430, Villetaneuse, France
[2] Nanyang Technological University, Singapore
[3] Singapore University of Technology and Design, Singapore
[4] School of Computing, National University of Singapore
[5] Temasek Laboratories, National University of Singapore

Abstract. Real-time systems are often hard to control, due to their complicated structures, quantitative time factors and even unknown delays. We present here PSyHCoS, a tool for analyzing parametric real-time systems specified using the hierarchical modeling language PSTCSP. PSyHCoS supports several algorithms for parameter synthesis and model checking, as well as state space reduction techniques. Its architecture favors reusability in terms of syntax, semantics, and algorithms. It comes with a friendly user interface that can be used to edit, simulate and verify PSTCSP models. Experiments show its efficiency and applicability.

1 Introduction

Ensuring the correctness of safety-critical systems, involving complex data structures with timing requirements, is crucial. The correctness of such real-time systems usually depends on the values of timing delays. Checking the correctness for one particular value for each delay is usually not sufficient for two reasons. Firstly, values for the delays are not always known, and one may precisely want to *find* some values for which the system behaves well. Secondly, even if the system is proved to be correct for a reference set of values, one has no guarantee that the correctness holds for other values around the reference ones. This is known as the *robustness* (see, e.g., [11]) of the system. Often, the engineer knows a correct reference valuation, but testing the correctness of the system for many values around it can turn very costly. Hence, it is interesting to consider the delays as unknown constants, or *parameters*, and synthesize constraints on these parameters to guarantee the correct behavior.

In this work, we present PSyHCoS (Parameter SYnthesis for Hierarchical COncurrent Systems), which supports editing, simulating, parameter synthesis and parametric model checking for Parametric Stateful Timed CSP (PSTCSP) [3]. PSyHCoS is a self-contained toolkit with extensible architecture design. To

⋆ This work is mainly supported by the TRF Project Grant No. R394-000-063-23 and the Seed Project Grant No. R394-000-068-232 from Temasek Lab@National University of Singapore.

N. Sharygina and H. Veith (Eds.): CAV 2013, LNCS 8044, pp. 984–989, 2013.

the best of our knowledge, PSyHCoS is the first tool that synthesizes timing parameters for real-time systems handling both hierarchy and concurrency.

The language PSTCSP offers an intuitive syntax for modeling hierarchical real-time systems involving shared variables, complex data structures, and user defined programs. PSTCSP (that is a parametric extension of Stateful Timed CSP [12]) is a process algebra with syntax for specifying concurrency (including conditional, general, external, internal choices, etc.) and timing requirements such as `Wait`$[d]$, `timeout`$[d]$, `within`$[d]$ and `deadline`$[d]$, where d can be a constant or a timing parameter. The expressiveness of PSTCSP is incomparable[1] with Parametric Timed Automata (PTA), which are an extension of finite state automata with clocks (variables increasing linearly) and parameters. Different from PTA, clocks in PSTCSP are *implicit* and dynamically created during the execution, thus avoiding the designer to write clocks constraints manually, which is error-prone. Another advantage of PSTCSP over PTA is the ability to easily define hierarchical systems, where sub-systems can be defined independently. Many systems can be designed more intuitively using hierarchy, and it may allow one to handle refinement as well as closed ("black box" or "gray box") systems.

Related Work. UPPAAL [10] is a tool for verifying (extensions of) timed automata. UPPAAL does not handle parameter synthesis and, although an extension performs parametric model checking [6], the model remains non-parametric. Some hierarchical extensions were considered, with limited tool support (e.g., [8,7]).

In [9], the process algebra ACSR-VP is used to synthesize timing constraints such that a real-time system is schedulable. PSyHCoS shares some principles with this approach, but does not limit to scheduling.

IMITATOR [2] is a tool performing parameter synthesis for PTA extended with stopwatches. Although IMITATOR has been extensively used, in particular for verifying models of industrial circuits, it does not feature any GUI nor simulation facilities and is limited to the inverse method (see Section 2). Last but not least, PSyHCoS can natively handle hierarchy whereas IMITATOR cannot.

2 Parameter Synthesis Made Easy

PSyHCoS offers a complete GUI for the design, simulation and verification of PSTCSP models. It comes with a user friendly editing environment (multi-document, multi-language interface, and advanced syntax editing features) for composing models. PSyHCoS also features a simulator that can be used for interactively and visually simulating system behaviors by random simulation, user-guided step-by-step simulation, complete state graph generation, trace playback, etc. Screenshots of the interface are available in PSyHCoS's Web page [1].

Among the verification algorithms, PSyHCoS first implements the inverse method *IM* initially defined for PTA [4] and extended to PSTCSP [3]. *IM* takes as input a PSTCSP model as well as a reference valuation π for all the parameters; it synthesizes a convex constraint K, that guarantees the same time-abstract behavior (sequences of actions) as for π. A major advantage is that K gives a

[1] Precisely, PSTCSP is equivalent to parametric closed timed ϵ-automata (see [3]).

quantitative measure of the robustness of the system w.r.t. variations of the timing delays. In particular, all linear time properties that hold for π also hold for any valuation in K. Parameter synthesis for PSTCSP has been proved to be undecidable [3] (as for PTA), and we were able to build examples on purpose for which *IM* does not terminate. However, for all practical case studies we considered, PSyHCoS does terminate. Exhibiting subclasses of PSTCSP for which termination of *IM* is guaranteed is the subject of ongoing work.

A full reachability algorithm *reachAll* is also implemented in PSyHCoS, in order to compare optimization techniques (see Table 1). Other classical model checking algorithms (such as LTL, deadlock freeness, or refinement checking) are also available. Data structures and functions can be written using the programming languages like C# and used seamlessly in the PSTCSP models.

We use Fischer's mutual exclusion algorithm to show PSyHCoS's intuitive modeling facilities. This hierarchical process (starting from `Fischer`) uses 2 timing parameters (`Delta` and `Epsilon`) and 2 variables. The `turn` variable indicates which process attempted to access the critical section most recently. The `counter` variable counts the number of processes accessing the critical section.

```
1  #define N 3;
2  #define Idle -1;
3  var turn = Idle;
4  var counter = 0;
5  parameter Delta;
6  parameter Epsilon;
7
8  proc(i) = ifb(turn == Idle) { Active(i) };
9  Active(i)=((update.i{turn=i} -> Wait[Epsilon])within[Delta]);
10    if (turn == i) {
11       cs.i{counter++} -> exit.i{counter--;turn=Idle}->proc(i)
12    } else {
13       proc(i)};
14 Fischer = ||| i:{0..N-1}@proc(i);
15
16 #synthesize Fischer with Delta = 3, Epsilon = 4;
```

`N` is a constant representing the number of processes. The parallel composition (line 14) automatically creates `N` processes in parallel. Process `proc(i)` models a process with a unique integer identify `i`. If `turn` is `Idle` (i.e., no other process is attempting), `proc(i)` behaves as specified by `Active(i)`. In `Active(i)`, `turn` is first set to `i` (i.e., the `i`th process is now attempting) by action `update.i`. Note that `update.i` must occur within `Delta` time units (captured by `within[Delta]`). Next, the process idles for `Epsilon` time units. It then checks if `turn` is still `i`. If so, it enters the critical section and leaves later. Otherwise, it restarts from the beginning.

A classical parameter synthesis problem is to find values for `Delta` and `Epsilon` such that mutual exclusion is guaranteed. This is achieved by calling the inverse method (at the last line) with `Delta=3` and `Epsilon=4` as a reference valuation.

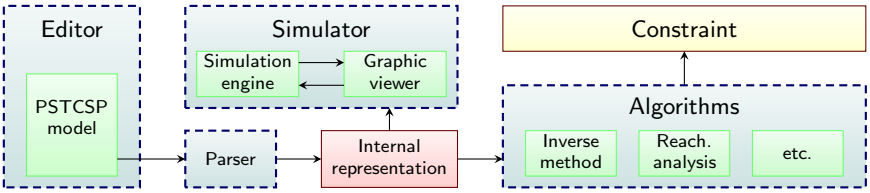

Fig. 1. Architecture of PSyHCoS

The constraint synthesized by PSyHCoS is `Delta` $<$ `Epsilon`, viz., the weakest (i.e., best) constraint known to guarantee mutual exclusion.

3 An Architecture Favoring Reusability

PSyHCoS is implemented in C#, based on Microsoft .NET framework, and uses the PPL library [5] for solving the parameter constraints. Sources, binaries, user manual and case studies are available in [1].

Reusability. The architecture of PSyHCoS is given in Fig. 1; it is fully object-oriented to favor reusability and ease the addition of new features. Each semantic rule of [3] is implemented in a different class, the methods of which are called every time the rule is applied. As a consequence, adding a new syntactic construct and its associated semantic rule simply requires one to add a new class to implement the semantics of the new syntax, and add a new line to the parser grammar file. Similarly, all algorithms are implemented in different classes. Although some algorithms are of course specific to PSTCSP, it is also possible to add algorithms that only depend on labeled transitions systems (such as LTL-checking, deadlock freeness, etc.). Such algorithms could be imported at no cost from existing model checking algorithms operating on LTS, e.g., Bogor, LTSA or the PAT model checking library.

Internal representation. The semantics of PSTCSP is defined as a labeled transition system, where the states in the transition system consist of a process and a constraint on clocks and parameters [3]. Each state is implemented as a pair (process id, constraint id), both coded as a string. Although some processing is needed each time a new state is computed, the constraint equality test reduces to string equality, which is faster than other representations. Furthermore, a string format is flexible – which is interesting as we are dealing with hierarchical systems so that different states may have very different system architecture.

Optimization. PSyHCoS implements a state-space reduction technique, that merges equivalent states in PSTCSP [3]. In the best case, this leads to an exponential diminution of the number of states, but at the cost of several nontrivial operations. The optimized version of *reachAll* (resp. *IM*) is denoted by *reachAll+* (resp. *IM+*). Both approaches are implemented so that users can choose.

Table 1. Application of algorithms for parameter synthesis using PSyHCoS

Case study	$\|U\|$	*reachAll* $\|S\|$	$\|T\|$	$\|X\|$	t	*reachAll+* $\|S\|$	$\|T\|$	$\|X\|$	t	*IM* $\|S\|$	$\|X\|$	t	*IM+* $\|S\|$	$\|X\|$	t
Bridge	4	-	-	-	M.O.	-	-	-	M.O.	2.8k	2	253	2.8k	2	455
Fischer_4	2	-	-	-	M.O.	-	-	-	M.O.	11k	4	41.9	2k	4	8.65
Fischer_5	2	-	-	-	M.O.	-	-	-	M.O.	133k	5	1176	13k	5	84.5
Fischer_6	2	-	-	-	M.O.	-	-	-	M.O.	-	-	M.O.	86k	6	1144
Jobshop	8	14k	20k	2	21.0	12k	17k	2	18.1	1112	2	17.1	877	2	22.8
RCS_5	4	5.6k	7.2k	4	10.5	5.6k	7.2k	4	9.54	5.6k	4	7.83	5.6k	4	16.7
RCS_6	4	34k	43k	4	91.7	34k	43k	4	54.5	34k	4	60.4	34k	4	91.3
TrAHV	6	7.2k	13k	6	14.2	7.2k	13k	6	15.8	227	6	0.555	227	6	0.655

4 Experiments and Discussion

We applied PSyHCoS to synthesize parameters for real-time systems ranging from classical concurrent algorithms to real world problems. In Table 1, we list the example names, the number $|U|$ of parameters and, for each algorithm, the number $|S|$ (resp. $|T|$) of states (resp. transitions), the maximum number $|X|$ of clocks, and the computation time t in seconds on a Windows XP 32 bits computer with an Intel Quad Core 2.4 GHz processor and 4 GB memory.

Bridge is a bridge crossing problem for 4 persons within 17 time units. Fischer_i is the mutual exclusion protocol for i processes. Jobshop is a scheduling problem. TrAHV is a classical train example for PTA. RCS_i is a railway control system with i trains. The reference valuation used for *IM* either is the standard valuation for the considered problem (Bridge, Jobshop, RCS_i, TrAHV) or has been computed in order to satisfy a well-known constraint of good behavior (Fischer_i).

When *reachAll* terminates, we can apply classical model checking algorithms: e.g., we checked that all models are deadlock-free (except Jobshop[2]). When *reachAll* does not terminate (Bridge, Fischer), *IM* is interesting because it synthesizes constraints despite an infinite set of reachable states; and when *reachAll* terminates slowly (TrAHV), *IM* may synthesize constraints much quicker.

The constraint output has several advantages. Firstly, it synthesizes values for which the system behaves well. Secondly, it gives a criterion of robustness to the system, by defining a safety domain around each parameter. Thirdly, it can happen that the constraint is *True* (e.g., RCS_i for all i). In this case, one can safely *refine* the model by removing all timing constructs (`Wait`, `deadline`, etc.). Although this might be checked using refinement techniques for one particular parameter valuation, we prove it here for *any* parameter valuation.

Also observe that, when *IM+* indeed reduces the number of states, it is much more efficient than *IM*, not only w.r.t. memory, but also w.r.t. time. However, with no surprise, when no state duplication is met (e.g., Bridge), the computation time is greater. Although reducing this computation is a subject of ongoing work, we do not consider it as a significant drawback: parameter synthesis' largest

[2] Jobshop is an acylic scheduling problem, where tasks should execute only once. Hence, once all actions have been performed, the system ends with a deadlock.

limitations are usually non-termination and memory saturation. Slower analyses for some case studies (up to +80% for Bridge) are acceptable when others benefit from a dramatic memory (and time) reduction (-90% for Fischer_5), allowing parameter synthesis even when *IM* goes out of memory (Fischer_6).

Comparison with IMITATOR. A comparison with IMITATOR (using the same machine with Ubuntu 11.10 64 bits) turned inaccurate. Indeed, the (manual) translation of models from PTSCP to PTA (and conversely) is difficult: in all cases, the tool for which the model was initially designed performs much better than the tool that runs on a translated model. For example, Jobshop (8.96 s) and TrAHV (0.097 s) are quicker on IMITATOR, for which they are designed. Conversely, IMITATOR does not terminate for Fischer_i for all i because of the explicit clock representation in PTA, whereas the implicit clocks in PSTCSP prevent this. Other models (Bridge, RCS_i) are too large to be manually translated. An automated efficient translation mechanism, that could ease such a comparison, is a subject of future work. Nevertheless, some features specific to PSTCSP, such as hierarchy, data structures, and implicit clocks, would be lost in any case by the translation.

References

1. PSyHCoS page, `http://lipn.univ-paris13.fr/~andre/software/PSyHCoS/`
2. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 33–36. Springer, Heidelberg (2012)
3. André, É., Liu, Y., Sun, J., Dong, J.S.: Parameter synthesis for hierarchical concurrent real-time systems. In: ICECCS 2012, pp. 253–262 (2012)
4. André, É., Soulat, R.: The Inverse Method. ISTE Ltd and John Wiley & Sons Inc. (2013)
5. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72(1-2), 3–21 (2008)
6. Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Beyond liveness: Efficient parameter synthesis for time bounded liveness. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 81–94. Springer, Heidelberg (2005)
7. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: ECDAR: An environment for compositional design and analysis of real time systems. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 365–370. Springer, Heidelberg (2010)
8. Dong, J.S., Hao, P., Qin, S., Sun, J., Yi, W.: Timed automata patterns. IEEE Transactions on Software Engineering 34(6), 844–859 (2008)
9. Kwak, H.-H., Lee, I., Sokolsky, O.: Parametric approach to the specification and analysis of real-time system designs based on ACSR-VP. Electronic Notes in Theoretical Computer Science 25, 38–49 (1999)
10. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
11. Markey, N.: Robustness in real-time systems. In: SIES 2011, pp. 28–34. IEEE Computer Society Press (2011)
12. Sun, J., Liu, Y., Dong, J.S., Liu, Y., Shi, L., André, É.: Modeling and verifying hierarchical real-time systems using Stateful Timed CSP. ACM Transactions on Software Engineering and Methodology 22(1), 3.1–3.29 (2013)