Information and Computation 262 (2018) 311–328

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Distributed synthesis for parameterized temporal logics ☆

Swen Jacobs *, Leander Tentrup, Martin Zimmermann

Reactive Systems Group, Saarland University, 66123 Saarbrücken, Germany

ARTICLE INFO

Article history:
Received 21 April 2017

Keywords:
Distributed synthesis
Distributed realizability
Incomplete information
Parametric linear temporal logic
Parametric linear dynamic logic

ABSTRACT

We consider the synthesis of distributed implementations for specifications in parameterized temporal logics such as PROMPT–LTL, which extends LTL by temporal operators equipped with parameters that bound their scope. For single process synthesis, it is well-established that such parametric extensions do not increase worst-case complexities. For synchronous distributed systems, we show that, despite being more powerful, the realizability problem for PROMPT–LTL is not harder than its LTL counterpart. For asynchronous systems, we have to express scheduling assumptions and therefore consider an assume-guarantee synthesis problem. As asynchronous distributed synthesis is already undecidable for LTL, we give a semi-decision procedure for the PROMPT–LTL assume-guarantee synthesis problem based on bounded synthesis. Finally, we show that our results extend to the stronger logics PLTL and PLDL.

1. Introduction

Linear Temporal Logic [1] (LTL) is the most prominent specification language for reactive systems and the basis for industrial languages like ForSpec [2] and PSL [3]. Its advantages include a compact variable-free syntax and intuitive semantics as well as the exponential compilation property, which explains its attractive algorithmic properties: every LTL formula can be translated into an equivalent Büchi automaton of exponential size [4]. This yields a polynomial space model checking algorithm and a doubly-exponential time algorithm for solving two-player games. Such games solve the monolithic LTL synthesis problem: given a specification, construct a correct-by-design implementation.

However, LTL lacks the ability to express timing constraints. For example, the request-response property $\mathbf{G}(req \rightarrow \mathbf{F}\, resp)$ requires that every request *req* is eventually responded to by a *resp*. It is satisfied even if the waiting times between requests and responses diverge, i.e., it is impossible to require that requests are granted within a fixed, but arbitrary, amount of time. While it is possible to encode an a-priori fixed bound for an eventually into LTL, this requires prior knowledge of the system's granularity and incurs a blow-up when translated to automata, and is thus considered impractical.

To overcome this shortcoming of LTL, Alur et al. introduced parametric LTL (PLTL) [5], which extends LTL with parameterized operators of the form $\mathbf{F}_{\leq x}$ and $\mathbf{G}_{\leq y}$, where x and y are variables. The formula $\mathbf{G}(req \rightarrow \mathbf{F}_{\leq x}\, resp)$ expresses that every request is answered within an arbitrary, but fixed, number of steps $\alpha(x)$. Here, α is a variable valuation, a mapping of variables to natural numbers. Typically, one is interested in whether a PLTL formula is satisfied with respect to some vari-

☆ Supported by the projects ASDPS (JA 2357/2–1) and TriCS (ZI 1516/1–1) of the German Research Foundation (DFG) and by the grant OSARES (No. 683300) of the European Research Council (ERC).

* Corresponding author.
E-mail addresses: jacobs@react.uni-saarland.de (S. Jacobs), tentrup@react.uni-saarland.de (L. Tentrup), zimmermann@react.uni-saarland.de (M. Zimmermann).

https://doi.org/10.1016/j.ic.2018.09.009
0890-5401/

able valuation, e.g., model checking a transition system $\mathcal{S}$ against a PLTL specification φ amounts to determining whether there is an α such that every trace of $\mathcal{S}$ satisfies φ with respect to α. Alur et al. [5] showed that the PLTL model checking problem is PSPACE-complete. Due to monotonicity of the parameterized operators, one can assume that all variables y in parameterized always operators $\mathbf{G}_{\leq y}$ are mapped to zero, as variable valuations are quantified existentially in the problem statements. Dually, again due to monotonicity, one can assume that all variables x in parameterized eventually operators $\mathbf{F}_{\leq x}$ are mapped to the same value, namely the maximum of the bounds. Thus, in many cases the parameterized always operators and different variables for parameterized eventually operators are not necessary.

Motivated by this, Kupferman et al. introduced PROMPT–LTL [6], which can be seen as the fragment of PLTL without the parameterized always operator and with a single bound k for the parameterized eventually operators. They proved that PROMPT–LTL model checking is PSPACE-complete and solving PROMPT–LTL games is 2EXPTIME-complete, i.e., not harder than LTL games. While the results of Alur et al. rely on involved pumping arguments, the results of Kupferman et al. are all based on the so-called alternating color technique, which basically allows to reduce PROMPT–LTL to LTL.

Intuitively, one introduces a new proposition that is thought of as coloring traces of a system. Then, one replaces each parameterized eventually operator $\mathbf{F}_{\leq x}\varphi$ by an LTL formula requiring φ to hold within at most one color change. If the distance between color changes is bounded from above, then satisfaction of the rewritten formula implies the existence of a bound k for the bounded eventually operators such that the original formula is satisfied with respect to k. Dually, if the distance between color changes is bounded from below, then the other implication holds: the original PROMPT–LTL formula implies the rewritten LTL formula.

When applying this equivalence, one has to specify how the truth values for the new atomic proposition coloring the traces are determined. In a game setting (in particular in synthesis), the player who aims to satisfy the PROMPT–LTL formula determines these truth values and is required to change colors infinitely often. Then, a finite-state strategy automatically ensures an upper bound on the distance between color changes.

Later, the result on PROMPT–LTL games was extended to PLTL games [7], relying on the monotonicity properties explained above and an application of the alternating color technique. These results show that adding parameters to LTL does not increase the asymptotic complexity of the model checking and the game-solving problem, which is still true for even more expressive logics like Parametric Linear Dynamic Logic (PLDL) [8] and PLTL and PLDL with costs [9]. The former logic is an extension of PLTL with the full expressiveness of the ω-regular languages. The latter logics are evaluated in weighted systems and generalize PLTL and PLDL by bounding the parameterized operators in the accumulated weight instead of bounding them in time.

The synthesis problems mentioned above assume a setting of complete information, i.e., every part of the system has a complete view on the system as a whole. However, this setting is unrealistic in distributed systems. Based on this observation, *distributed synthesis* is defined as the problem of synthesizing multiple components with incomplete information. Since there are specifications that are not implementable, one differentiates synthesis from the corresponding decision problem, i.e., the *realizability* problem of a formal specification. We focus on the latter, but note that typically algorithms for the realizability problem also solve the synthesis problem, as they rely on constructing implementations to prove realizability. This also holds in our work here.

The realizability problem for distributed systems dates back to work of Pnueli and Rosner in the early nineties [10]. They showed that the realizability problem for LTL becomes undecidable already for the simple architecture of two processes with pairwise different inputs. In subsequent work, it was shown that certain classes of architectures, like pipelines and rings, can still be synthesized automatically [11,12]. Later, a complete characterization of the architectures for which the realizability problem is decidable was given by Finkbeiner and Schewe by the *information fork* criterion [13]. Intuitively, an architecture contains an information fork if there is an information flow from the environment to two different processes where the information to one process is hidden from the other and vice versa. The distributed realizability problem is decidable exactly for those architectures without an information fork. Beyond decidability results, semi-decision procedures like bounded synthesis [14] give an architecture-independent synthesis method that is particularly well-suited for finding small-sized implementations. Bounded synthesis searches for finite-state implementations of a fixed size by encoding the problem as a constraint system in a decidable first-order theory. In case of a positive answer, the result is returned, otherwise the bound is increased. If there is an upper bound on the size of a finite-state implementation, then bounded synthesis is a complete decision procedure, as it can be stopped if the upper bound is reached without a positive answer. If there is no such upper bound, it is indeed a semi-decision procedure that finds an implementation if one exists, but runs forever otherwise.

1.1. Our contributions

As mentioned above, one can add parameters to LTL for free: the complexity of the model checking problem and of solving infinite games does not increase. This raises the question whether this is also true for distributed realizability of parametric temporal logics. For synchronous systems, we can answer this question affirmatively. For every class of architectures with decidable LTL realizability, the PROMPT–LTL realizability problem is decidable, too. To show this, we apply the alternating color technique [6] to reduce the distributed realizability problem of PROMPT–LTL to the one of LTL: one can again add parameterized operators to LTL for free. To prove this result, we add a new process whose only task it is to determine a coloring with the fresh proposition. By ensuring that the new process does not introduce an information fork we obtain decidability for the same class of architectures as for LTL.