The *sat* problem of mu-calculus over Petri Nets

Feuillade Guillaume

INRIA, France, `guillaume.feuillade@irisa.fr`

SUPERVISOR Pinchinat Sophie

KEYWORDS: Synthesis, Petri nets, concurrency, control, mu-calculus

Abstract

We study the decidability of the synthesis of unlabeled Petri Net from logical specifications. We discuss Petri Net synthesis methods and results. We present a proof for the undecidability of the synthesis problem for a sentence of the mu-calculus.

1 Introduction

In the area of automatic system validation, many formal languages to express properties and many formal representations of systems have been proposed. Given both a language of specifications $\mathcal{L}$ and a class of models $\mathcal{M}$, the automatic validation can be performed in two ways : *a posteriori* and *a priori*. This introduces two theoretical problems.
The first one is: given a model M in the class $\mathcal{M}$ and a property ϕ expressed within $\mathcal{L}$, "does M verifies the property ϕ ?". This problem corresponds to *the model-checking problem of* $\mathcal{L}$ *over* $\mathcal{M}$ and is noted **Model-Checking($\mathcal{L}, \mathcal{M}$)**.
The second one is: given a property ϕ expressed within $\mathcal{L}$, "is there a model M in the class $\mathcal{M}$ such that M verifies ϕ ?". This problem corresponds to *the sat problem of* $\mathcal{L}$ *over* $\mathcal{M}$ and is noted **Sat($\mathcal{L}, \mathcal{M}$)**.
The model-checking problem is related to system verification : given a system or its abstraction and a set of required properties, it permits to validate or invalidate the system. The sat problem addresses two goals : the first one is the synthesis of a system or its abstraction from required properties, the second one is the synthesis of a controller to ensure that the combination of a given system and the controller follows some required properties. We consider these two problems with a particular logic, the mu-calculus, and recall results about model-checking and sat over finite regular processes. Since finite regular processes have flaws, we consider Petri Nets as class of models, and show that the synthesis problem in its generality is undecidable, but is decidable for specific restrictions over the specifications or over the models.

2 Mu-calculus

The mu-calculus of [7], or Hennessy-Milner logic with fix-points, is a popular logic for expressing temporal properties of reactive systems. The mu-calculus, or L_μ is not a state-based logic, as opposed to MSO for example, and expresses properties over the execution tree for its branching-time version, or over execution words for its linear-time version. One consequence is an underlying equivalence of models : bisimulation. Basically, two systems are equivalents w.r.t. bisimulation if and only if they have the same execution tree. Mu-calculus sentences does not distinguish two systems that are bisimilar. If M_1 and M_2 are two models equivalents w.r.t. bisimulation, and ϕ is a mu-calculus sentence then M_1 satisfies ϕ if and only if M_2 satisfies ϕ.

The theory of L_μ is related to alternating tree automata with parity acceptance [1, 2]. In fact, each mu-calculus sentence is equivalent to an alternating tree automaton and reciprocally. For each sentence ϕ of the mu-calculus, there exists an alternating tree automaton $\mathcal{A}_\phi$ such that the models in $\mathcal{M}$ that satisfies ϕ are exactly the models accepted by the automaton $\mathcal{A}_\phi$; and for each alternating tree automaton $\mathcal{A}$, there exists a sentence ϕ of the mu-calculus such that the models accepted by the automaton $\mathcal{A}$ are exactly the models that satisfies ϕ.
Methods to solve **Model-Checking**$(L_\mu, \mathcal{M})$ and **Sat**$(L_\mu, \mathcal{M})$, for some classes of models $\mathcal{M}$, rely on tree automatons and elements from games theory. In particular, when $\mathcal{M}$ is the set of finite regular processes $\mathcal{R}$ (or equivalently finite Kripke structures), a finite two-players parity game can be built according to [2] from the specification to solve **Model-Checking**$(L_\mu, \mathcal{R})$ and **Sat**$(L_\mu, \mathcal{R})$. The first step for solving both problems for a sentence ϕ of L_μ is to transform the alternating tree automaton into a non-deterministic tree automaton with parity acceptation according to the simulation theorem [1].
An instance of **Sat**$(L_\mu, \mathcal{R})$ is a sentence ϕ of L_μ. From the non-deterministic tree automaton obtained from ϕ, a finite two players parity game for this instance of the $sat(L_\mu, \mathcal{R})$ problem can be built, which is determined [5] ; this means that either player 0 or player 1 has a winning strategy. In order to compute a solution for this instance of **Sat**$(L_\mu, \mathcal{R})$, one must found a winning strategy for player 0 and build, when possible, a model according to the strategy found. Fortunately, when the class of models is $\mathcal{R}$, there is two crucial properties that enable such a construction. The first one is : when player 0 has a winning strategy, then player 0 has a positional winning strategy. The second one is : when player 0 has a positional winning strategy, the construction of a finite regular process according to this strategy is effective. With these two properties, **Sat**$(L_\mu, \mathcal{R})$ is decidable, and the construction of a model satisfying a given sentence ϕ of L_μ is effective.
Similarly, an instance of **Model-Checking**$(L_\mu, \mathcal{M})$ is a sentence ϕ of L_μ and a model M within the class $\mathcal{R}$. From the non-deterministic tree automaton obtained from ϕ and the finite regular process M, a finite two player parity game can be built. This game is determined and $\mathcal{M}$ satisfies ϕ if and only if player 0 has a winning strategy in this game. Thus solving an instance of **Model-Checking**$(L_\mu, \mathcal{M})$ is equivalent to the research of a winning strategy for player 0. If there is one, the answer is "yes", else player 1 has a win-

ning strategy and the answer is "no". Hence the model-checking of L_μ over finite regular processes is effective.

3 Labeled and unlabeled Petri Nets

While regular processes are purely sequential, there exists others class of models which are concurrent. We focus on Petri Nets and subclasses of Petri Nets [8] ; they were first introduced by Carl Adam Petri and are concurrent systems based on the notion of places, transitions, and resources. Let Σ be an alphabet.

A labeled Petri Net is given a labeling function ranging from the places of the net to elements of Σ. We define the execution tree of a labeled Petri Net as the unfolding of its marking graph. Since two transitions may have the same image for the labeling function, labeled Petri Nets are non-deterministic systems. From a finite regular process, an unlabeled Petri Net can be built which is bisimilar to the regular process : the places are the states of the process while the transitions are the arcs of the process, and the labeling function associates each arc label to the corresponding transition of the net ; finally the place corresponding to the initial state of the process is given one unique resource. Thus the class of unlabeled Petri Net is a superclass of $\mathcal{R}$.

An unlabeled Petri Net is a net whose transitions are the elements of Σ (or labeled injectively over Σ). Unlabeled Petri Nets are deterministic systems. The execution tree is defined as the unfolding of the marking graph of the net. It is possible to built a distributed implementation of an unlabeled Petri Net exploiting concurrence. As a class of model, unlabeled petri nets, noted $\mathcal{UPN}$, is neither a subclass of $\mathcal{R}$ nor a superclass of $\mathcal{R}$.

4 Petri Net Synthesis

It has been shown in [3] that it is possible to determine if there is a net in $\mathcal{UPN}$ whose language is equal to a given prefix closed regular language, and to build it. When this net does not exist, it is possible to build the net with minimal language, in the sense of the set inclusion, whose language contains the given prefix-closed regular language. Furthermore, Petri Net synthesis is effective for more complex structures [4] : automatic graphs and automatic specifications.

The first consequence is that from a finite regular system, it is possible to determine if there is an unlabeled Petri Net bisimilar to the regular system, and to synthesize it. Thus, it is possible to solve an instance of $\mathbf{Sat}(L_\mu, \mathcal{R})$ to obtain a regular process, and to synthesize from this process, when possible, a net in $\mathcal{UPN}$ which is a solution of the same problem over the class $\mathcal{UPN}$ of models. However, since $\mathcal{R}$ is not a subset of $\mathcal{UPN}$, the synthesis is not always effective ; moreover, the sequential system may not be a proper choice for this two step synthesis of net. That is to say that between all the finite states regular models satisfying the mu-calculus sentence, the chosen one has no guarantees of being the proper intermediate model for an $\mathcal{UPN}$-synthesis.

5 Sat problem of L_μ for $\mathcal{UPN}$

We show that **Sat**(L_μ,$\mathcal{UPN}$) is undecidable, as is **Model-Checking**(L_μ,$\mathcal{UPN}$). The model checking problem for Petri nets is known to be undecidable since 1994 [6], but the relationship between sat and model-checking problems has not been established yet. Both problems are related to the halt problem for Minsky machines : the Petri Nets provide the counters behaviors while branching time logic provide regular reachability and zero test on the counters.

For the model checking problem, the undecidability proof relies on simulating a Minsky machine with a Petri Net and a sentence of the mu-calculus which one is satisfied if and only if the Minsky machine eventually halts.
The sat problem can be prooved undecidable a similar way.

Theorem 1 *the problem* **Sat**(L_μ,$\mathcal{UPN}$) *is undecidable*

For this problem, the undecidability proof is a reduction of the problem of the research of an initial configuration for the Minsky machine to halt for, which is undecidable. The mu-calculus sentence provides reachability, according to the machine's structure, zero test ; moreover this sentence is designed to force the structure of the family of Petri Nets which marking graph satisfies the sentence. This avoids trivial solutions and ensures that if there is a Net which marking graph satisfies the sentence then there is another one, which marking graph satisfies the sentence too, and providing the expected counter behavior. The structure of solutions is forced by modifying the zero-test formula, using a greatest fix point ensuring that the test is performed only by a place behaving like the counter of a Minsky machine. This way, the sat problem associated with this sentence has a solution (a Petri Net) if and only if there exists an initial configuration for the Minsky machine to halt for ; thus this proves that the sat problem on Petri Nets is undecidable. Note that this result require the use of the first level of alternation of the mu-calculus.

In contrast with this undecidability result, synthesis of unlabeled 1-safe Petri nets is effective for the mu-calculus and even for MSO [9]. This result relies on the fact that the possible structures and markings of unlabeled 1-safe Petri nets are finite.
However, we show that the problems called marking-synthesis problem and consisting in finding an initial marking for a given net structure in order to satisfy a given mu-calculus sentence are also undecidable when these too conditions are satisfied :

1. 1 level of fix-point alternation is permitted,
2. the class of models (subclass of $\mathcal{UPN}$), is larger or equal to marked graphs.

6 Conclusion and perspectives

We showed that the sat problem over $\mathcal{UPN}$ is undecidable for the whole mu-calculus. The next issue is to find a fragment of the mu-calculus for which the synthesis is effective. We defined a syntactic restriction of the mu-calculus : the conjunctive-nu-calculus, noted L_ν. The conjunctive-nu-calculus is obtained from the mu-calculus by removing the following operators : not, or, the least fix-point. We then define a subclass of tree automata, called modal automata, having the same relation with the conjunctive-nu-calculus than alternating tree automata with mu-calculus. These specifications differ from automatic specifications in the sense that states of the system have to be guessed in order to synthesize a net from $\mathcal{UPN}$, while all the states of an automatic specification are reachable. The problem $\mathbf{sat}(L_\nu,\mathcal{UPN})$ has been solved for some structural restrictions on modal automata, mainly about the cycles in the automaton (or equivalently the fix-points in the formula) : for example, the variable of a fix-point should occur only once in the associated sub-formula. However the problem remains open in its generality. We expect to extend these structural restriction and to characterize it syntactically in L_ν. The further step would be to express the solutions in term of strategies in the associated parity game, and to study the properties of these strategies.

References

[1] A. Arnold and D. Niwinski. *Rudiments of mu-calculus.* North-Holland, 2001.

[2] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *to appear in Theoretical Computer Science*, 2003.

[3] E. Badouel and P. Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer, 1999.

[4] E. Badouel and P. Darondeau. The petri net synthesis problem for automatic graphs. Technical Report 4661, INRIA Rennes, December 2002.

[5] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. *Proceedings 32nd Annual IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.

[6] J. Esparza. On the decidabilty of model checking for several mu-calculi and petri nets. In S. Tison, editor, *Proceedings of Trees in Algebra and Programming - CAAP '94, 19th International Colloquium 1994*, number 787 in Lecture Notes in Computer Science, pages 115–129, 1994.

[7] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[8] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–574, April 1989.

[9] K. Sunesen. Reasoning about reactive systems, 1998.