AlPiNA: An Algebraic Petri Net Analyzer*

Didier Buchs, Steve Hostettler, Alexis Marechal, and Matteo Risoldi

Software Modeling and Verification laboratory
University of Geneva, Route de Drize 7, CH-1227 Carouge, Switzerland
http://smv.unige.ch

Abstract. *AlPiNA* is a graphical editor and model checker for a class of high-level Petri nets called *Algebraic Petri Nets*. Its main purpose is to perform reachability checks on complex models. It performs symbolic model checking based on ΣDD, an efficient evolution in the Decision Diagrams field, using novel techniques such as *algebraic clustering* and *algebraic unfolding*. AlPiNA offers a user-friendly interface, and is easily extensible.

1 Introduction

This article introduces the AlPiNA model checking tool. AlPiNA allows checking reachability properties on Algebraic Petri Nets (APN) models, a class of High Level Petri Nets. It encodes state spaces symbolically as Decision Diagrams [6], which reduces memory consumption and computation time that are major obstacles to the practical use of model checking. Users can specify properties to verify using a dedicated language, and they can provide additional information on the model to improve model checking performance. In the current iteration of AlPiNA, we focus on reachability properties for several reasons – among others, the fact that many interesting properties can be expressed as reachability properties as proven in CPN Tools [7].

AlPiNA has two main goals. The first goal is improving model checking performance by leveraging the Decision Diagrams framework and the innovative concepts of *algebraic clustering* and *algebraic net unfolding*. Algebraic clustering reduces the memory footprint of state space calculation by semi-automatically decomposing the system in independent processes. Partial algebraic net unfolding allows reducing the complexity of the data type unfolding. The second goal of AlPiNA is coupling this high performance with a user friendly interface. The user can specify models and properties with a graphical and textual editor. We propose to separate the model and performance-related information. This gives the users a high-level view of the model, freeing them from the need to use low-level formalisms in a complex way.

The article is structured as follows. Section 2 quickly illustrates the theoretical foundations of AlPiNA. Section 3 describes the tool's architecture and shows some benchmarks. Finally, the tool's current status and perspectives are discussed.

* This project was partially funded by the COMEDIA project of the Hasler foundation, ManCom initiative project number 2107.

J. Esparza and R. Majumdar (Eds.): TACAS 2010, LNCS 6015, pp. 349–352, 2010.

2 Theoretical Foundations of AlPiNA

Concurrency and non-determinism are the major causes of exponential state space explosion [12]. This happens when model components have few causal dependencies with each other and therefore evolve almost independently. Because of the exponential nature of the model checking problem, the state space rapidly becomes intractable as the number of components increases. To overcome this, the state space encoding must have a lower complexity than the explicit enumeration of states. We extend the approach initiated by McMillan [4] called Symbolic Model Checking, which exploits maximal sharing of state elements. In APNs, values are instances of algebraic abstract data types (ADT), therefore they require a more powerful encoding of the state space than Binary Decision Diagrams [1]. Because of this, we defined an evolution of Decision Diagrams (DD) [6] called ΣDD [3].

Clusters (i.e. sets of states) maximize the sharing induced by encoding with DDs [6]. For example, all the places of a Petri net that represent a process and its resources are grouped together. In this case, the cluster is called a *topological cluster* [9] since it is solely based on the Petri net topology. In high-level Petri nets, because of the level of abstraction, places can represent *classes* of similar processes and resources. *Algebraic clusters* [2] allow the user to group process instances with their resources. AlPiNA automatically derives clustering from this grouping. The more independent the resulting clusters are, the more efficient the symbolic representation will be. In the best case, the memory consumption is logarithmic to the number of states.

Since AlPiNA uses APNs, it has to manipulate universally quantified variables. An interesting way of improving performance is to perform an *algebraic net unfolding* [2]. It instantiates the variables of the system in a pre-processing phase, before state space exploration. By doing this, it becomes possible to compile the model with bindings that satisfy the transition guards. Unfolding may significantly increase the speed of the state space construction when the data domains are finite or bounded. Still, it is not always possible or even desirable to perform unfolding for two reasons. The first reason is that a bound may be difficult to figure out: if the bound is too small, the validation becomes incorrect; if it is too large, unfolding may become very expensive and model checking itself intractable. The second reason is that sometimes it is useless to unfold a data domain if only a few of its values are effectively used.

To tackle this problem, we propose to perform *partial unfolding*, i.e. choosing only a subset of the domains. The choice whether a domain should be part of the unfolding is a trade-off between the possible speed gain and the cost of the unfolding itself. Its computational complexity is $O(n^c)$ where n is the size of the largest data domain and c the largest number of input arcs.

In AlPiNA we generate the state space using an algorithm called saturation [5]. The algorithm benefits from the clustering of the state space to *fire* all transitions local to a component before firing inter-component transitions. All the transitions local to a given cluster are only applied to the subset of the state space relevant to the cluster, avoiding superfluous computations. A detailed technical description of the encoding as well as the notion of *algebraic cluster* and *algebraic unfolding* has been given in [2].

3 Tool Description

AlPiNA's architecture can be seen in Fig. 1. AlPiNA is composed of a *Model Checker Engine* (1) and a *Graphical User Interface* (2) built on top of it.

AlPiNA's architecture can be seen in Fig. 1. AlPiNA is composed of the *Model Checker Engine* (1) and a *Graphical User Interface* (2) built on top of it.

The foundation of the *Model Checker Engine* is the symbolic representation offered by DD structures, as presented in the previous section. The first two layers of our engine refer to libraries that handle DD structures. The third layer is a bridge between the APN semantics and the underlying layers. It performs optimisations such as algebraic clustering and net unfolding. On top of the engine block, we find the property checker layer, that uses the state space generated by the previous layer to compute the properties satisfaction. These two layers communicate with the GUI block, they receive the models and return the generated state space and properties statisfaction results.

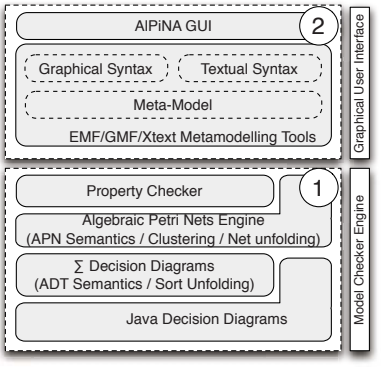

Fig. 1. Architecture Overview

		AlPiNA				Maria		Helena	
		Partial Unfold.		Total Unfold.					
Model Size	States #	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)	Mem (MB)	Time (s)
Distributed Database									
10	197E3	10	0.8	12.4	1.3	47	44.3	24	9
15	7.2E7	33	2.6	41	5.8	-	-	1.4E3	7.5E3
35	5.8E17	544	69.4	789	278	-	-	-	-
Dining Philosophers									
10	186E4			1.9	0.15	375	141	11	5
15	2.5E9			2.6	0.18	-	-	409	822
300	1.2E188			162	48.5	-	-	-	-
Slotted Ring									
5	53856			4.9	0.2	23	4.3	10	5
10	8.3E9			55.6	1.7	-	-	-	-
15	1.5E15			330	9.8	-	-	-	-
Leader Election									
15	399E4			27.7	1.4	795	361	107	142
50	1.7E21			702	76	-	-	-	-

Fig. 2. State space generation

The second block of the AlPiNA architecture is the GUI. We used the Eclipse Tools from the Eclipse Modeling Project (EMP) [8] to create a user friendly interface, following the MDA directives. The first layer is the metamodels specifications, created with EMF. With these metamodels, we created a graphical concrete syntax using GMF for the Petri Nets editor, and a textual concrete syntax using XText for the textual editors. This schema allows us to create an extensible and modular tool.

AlPiNA has good memory consumption and processing time as shown in Fig. 2. It outperforms by an order of magnitude two widely used high level Petri nets model checkers, Maria [10] and Helena [11]. This figure shows the results obtained for some well known examples in the model checking field. The "–" symbol indicates that a result could not be computed[1]. Every example shows that the techniques we present in this tool can produce excellent results when applicable. The distributed database example

[1] These benchmarks were computed using a 4 GB ram, 2.5 GHz Core 2 Duo Macbook Pro. The source code can be downloaded at http://alpina.unige.ch

shows also that *partial net unfolding* can give better results than total unfolding. The blank cells indicate that the test has not been run. Indeed, *partial net unfolding* is not useful when the algebra are to small, which is the case for the *Dining Philosophers*, the *Slotted Ring* and the *Leader election*.

4 Current Status and Perspectives

Compared to other high-level model checkers, AlPiNA has the advantage of treating state spaces larger by orders of magnitude while being user friendly. Users benefit from the efficiency based on the Decision Diagrams technology in a transparent manner. They can also easily specify *algebraic clustering* and *algebraic net unfolding* to improve model checking performance. Thanks to this, AlPiNA outperforms Maria and Helena when the model has strong concurrency.

All the features mentioned in this paper have been implemented in AlPiNA. A public release can be found at *http://alpina.unige.ch*. The tool has a user-friendly interface, taking full advantage of the EMF tools features. We are currently working on the next version which should bring modularity to the formalism and CTL support. Moreover, we will improve user guidance while defining the algebraic clustering.

References

1. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. Transactions on Computers C-35, 677–691 (1986)
2. Buchs, D., Hostettler, S.: Managing complexity in model checking with decision diagrams for algebraic petri net. In: Moldt, D. (ed.) Pre-proceedings of the International Workshop on Petri Nets and Software Engineering, pp. 255–271 (2009), http://smv.unige.ch/publications/pdfs/pnse09.pdf
3. Buchs, D., Hostettler, S.: Sigma Decision Diagrams: Toward efficient rewriting of sets of terms. In: Corradini, A. (ed.) TERMGRAPH 2009: Preliminary proceedings of the 5th International Workshop on Computing with Terms and Graphs, number TR-09-05 in TERMGRAPH workshops, Università di Pisa, pp. 18–32 (2009), http://smv.unige.ch/publications/pdfs/termgraph09.pdf
4. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf. Comput. 98(2), 142–170 (1992)
5. Ciardo, G., Lüttgen, G., Siminiceanu, R.: Efficient symbolic state-space construction for asynchronous systems. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 103–122. Springer, Heidelberg (2000)
6. Couvreur, J.-M., Thierry-Mieg, Y.: Hierarchical decision diagrams to exploit model structure. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 443–457. Springer, Heidelberg (2005)
7. CPN Group. CPN tools, http://wiki.daimi.au.dk/cpntools/cpntools.wiki
8. Eclipse. Eclipse modeling project, http://www.eclipse.org/modeling/
9. Hamez, A., Thierry-Mieg, Y., Kordon, F.: Hierarchical set decision diagrams and automatic saturation. In: Petri Nets, pp. 211–230 (2008)
10. Mäkelä, M.: Modular reachability analyzer, http://www.tcs.hut.fi/Software/maria/
11. Pajault, C., Evangelista, S.: High level net analyzer, http://helena.cnam.fr/
12. Valmari, A.: The state explosion problem. In: Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, London, UK, pp. 429–528. Springer, Heidelberg (1998)