# A Term Rewriting Approach to Analyze High Level Petri Nets

Conference Paper · July 2016

DOI: 10.1109/TASE.2016.11

5 authors, including:

Xudong He
Florida International University
152 PUBLICATIONS 1,269 CITATIONS
SEE PROFILE

Reng Zeng
Florida International University
13 PUBLICATIONS 57 CITATIONS
SEE PROFILE

Su Liu
Florida International University
7 PUBLICATIONS 37 CITATIONS
SEE PROFILE

# A Term Rewriting Approach to Analyze High Level Petri Nets

Xudong He*, Reng Zeng†, Su Liu†, Zhuo Sun*, Kyungmin Bae‡
*Florida International University, Miami, FL 33199, USA
†Citrix Systems Inc., Ft. Lauderdale, FL 33309, USA
‡SRI International, Menlo Park, CA 94025, USA

*Abstract*—**High level Petri nets (HLPNs) have been widely applied to model concurrent and distributed systems in computer science and many other engineering disciplines. However, due to the expressive power of HLPNs, they are difficult to analyze. In recent years, a variety of new analysis techniques based on model checking have been proposed to analyze high level Petri nets in addition to the traditional analysis techniques such as simulation and reachability (coverability) tree. These new analysis techniques include (1) developing tailored model checkers for particular types of HLPNs or (2) leveraging existing general model checkers through model translation where a HLPN is transformed into an equivalent form suitable for the target model checker. In this paper, we present a term rewriting approach to analyze a particular type of HLPNs – predicate transition nets (PrT nets). Our approach is completely automatic and implemented in our tool environment, where the frontend is PIPE+, a general graphical editor for creating PrT net models; and the backend is Maude, a well-known term rewriting system. We have applied our approach to the Mondex system – the 1st pilot project of verified software repository in the worldwide software verification grand challenge, and several well-known problems used in the annual model checking contest of Petri net tools. Our initial experimental results are encouraging and demonstrate the usefulness of the approach.**

## I. Introduction

High level Petri nets (HLPNs) have been widely applied to model concurrent and distributed systems in computer science and many other engineering disciplines. However, due to the expressive power of HLPNs, they are more difficult to analyze. In recent years, a variety of new analysis techniques based on model checking have been proposed to analyze high level Petri nets in addition to the traditional analysis techniques such as simulation and reachability (coverability) tree. These new analysis techniques include (1) developing tailored model checking algorithms for particular types of HLPNs ([1]), or (2) leveraging existing general model checkers through model translation where a HLPN is transformed into an equivalent form suitable for the target model checker ([2], [3], [4]). There are two main approaches in developing model checking algorithms for analyzing HLPNs. The first approach uses classical explicit or symbolic techniques [5] to represent and calculate state spaces of a given HLPN. For example, CPN Tools [6] supports explicit model checking of formulas written in a logic similar to computation tree logic (CTL) in Colored Petri nets (CPN – a type of HLPNs) [7], while AlPiNa [8] is a symbolic model checker based on decision diagrams for algebraic Petri nets (APN – a type of HLPNs). The second approach explores partial order semantics of HLPNs based on the net unfolding technique proposed in [9]. For example, in [1], an approach was developed to unfold the causal (partial order) behaviors into branching processes of M-net (a type of HLPNs), which was shown to be more effective than first expanding a HLPN into an equivalent low level Petri net (LLPN) and then unfolding the LLPN into low level branching processes through the greatest common divisor and mutual exclusion examples. However, it is not clear whether this approach has been built into a publicly available HLPN modeling and analysis tool. Several tools supporting direct model checking of HLPNs participated in the annual Petri net model checking contest [10], but had very limited success. An alternative way of analyzing HLPNs leverages existing well-known general purpose model checkers. A translation method is needed to generate a state transition system from a given HLPN. Model translations have been used and explored for many years [11]. The key of a translation method is to ensure the behavioral equivalence of the source model and target model. It is quite straightforward to obtain a state transition system from a given HLPN with the loss of true concurrency. Fortunately, safety and liveness properties are preserved from all known Petri net semantics – causal (partial order), maximal concurrency, interleaving set, and interleaving. In our prior work ([2], [3], [4]), we developed methods to translate PrT nets into the input models of several well-known model checkers including SMV [12] and SPIN [13]. SPIN has now been integrated into our PIPE+ tool [14] as well as SAMAT environment [15] as a backend model checking engine. Although tailored model checkers for HLPNs may take advantages of the unique features of the underlying HLPNs and perform specific optimizations to improve performance, they often lack the full-fledged features provided by the well-known general model checkers and are not easily adaptable. Furthermore, there are no convincing experiments yet to show tailored model checkers have performance advantages over using general model checkers. In this paper, we present a term rewriting approach to analyze a particular type of HLPNs – predicate transition nets (PrT nets). Our approach is completely automatic and implemented in our tool environment, where the frontend is PIPE+, a general graphical editor for creating PrT net models; and the backend is Maude, a well-known term rewriting system. We demonstrate the application of our approach through several sample systems, including the Mondex system [4] – the 1st pilot project of verified software repository in the worldwide software verification grand challenge, and several other well-known problems used in the annual model checking contest of Petri net tools.

## II. Predicate Transition Nets

In the following sections, we give a brief definition of PrT nets using the conventions in [16], which has a flavor of algebraic Petri nets [17] and is different from the original definition of PrT nets in [18].

### A. The Syntax and Static Semantics of PrT Nets

A PrT net is a tuple $(N, Spec, ins)$ where (1) $N = (P, T, F)$ is a net structure, in which (i) $P$ and $T$ are disjoint non-empty finite sets (the sets of places and transitions of $N$ respectively), (ii) $F \subseteq P \times T \cup T \times P$ is a flow relation (the arcs of $N$); (2) $Spec = (S, Op, Eq)$ is the underlying algebraic specification, and consists of a signature $\Sigma = (S, Op)$ and a set $Eq$ of $\Sigma$-equations. Signature $\Sigma$ includes a set of sorts $S$ and a family $Op = (Op_{s1,...,sn;s})$ of sorted operations for $s1, ..., sn, s \in S$. For $s \in S$, we use $Cons_s$ to denote $Op_{;s}$ (the 0-ary operation of sort $s$), i.e. the set of constant symbols of sort $s$. The $\Sigma$-equations in $Eq$ define the meanings and properties of operations in $Op$. $Spec$ is a meta-language to define the tokens, labels, and constraints of a PrT net. Tokens of a PrT net are ground terms of signature $\Sigma$, written $MCons_S$. The set of labels is denoted by $Label_S(X)$ ($X$ is the set of sorted variables disjoint with $Op$). Each label can be a simple variable or a set expression of the form $\{x_1, ..., x_n\}$. Constraints of a PrT net are a subset of first order logic formulas (where the domains of quantifiers are finite and any free variable in a constraint appears in the label of some connecting arc of the transition), and thus are essentially propositional logic formulas. The subset of first order logical formulas contains the $\Sigma$-terms of sort Bool over $X$, denoted as $Term_{(Op;Bool)}(X)$; (3) $ins = (\varphi, L, R, M_0)$ is a net inscription that associates a net element in $N$ with its denotation in $Spec$: (i) $\varphi : P \to \wp(S)$ is the data definition of $N$ and associates each place $p$ in $P$ with a subset of sorts in $S$. (ii) $L : F \to Label_S(X)$ is a sort-respecting labeling of PrT net. We use the following abbreviation in the following definitions: $\bar{L}(x, y) = L(x, y)$ if $(x, y) \in F$ or $\bar{L}(x, y) = \emptyset$ if $(x, y) \notin F$ (iii) $R : T \to Term_{(Op;Bool)}(X)$ is a well-defined constraining mapping, which associates each transition $t$ in $T$ with a first order logic formula defined in the underlying algebraic specification. Furthermore, the constraint of a transition defines the meaning of the transition. We use $Var(t)$ to denote variables appearing in $R(t)$. (iv) $M_0 : P \to MCons_S$ is a sort-respecting initial marking. The initial marking assigns a multiset of tokens to each place $p$ in $P$. A $\Sigma$-algebra of $Spec$ provides interpretations for the sorts and operations in $Spec$, and includes familiar sorts such as integer, Boolean, string, tuple, and set as well as their relevant operations and equations. In our tool environment, the $\Sigma$-algebra is instantiated with a subset of Java data types and their associated operations and laws.

### B. Dynamic Semantics of PrT Nets

The dynamic semantics of a PrT net concerns the behaviors defined by the transition firing sequences and the resulting marking sequences and reached marking sets.

Markings of a PrT net $N$ are mappings $M : P \to MCons_S$. A transition $t \in T$ is *enabled* under a marking $M$ if there is a substitution $\alpha$ such that $\forall p \in P.(\alpha(\bar{L}(p, t)) \subseteq M(p)) \wedge \alpha(R(t))$ is true, where $\alpha(e)$ denotes the result of instantiating the variables in expression $e$ with ground terms. An enabled transition under marking $M$ with substitution $\alpha$ can *fire*, which results in a new marking $M'$ defined by $\forall p \in P.(M'(p) = M(p) \setminus \alpha(\bar{L}(p, t)) \cup \alpha(\bar{L}(t, p)))$. We use $M[t/\alpha > M'$ to denote the firing of $t$ with substitution $\alpha$ under marking $M$, and $[M >$ to denote the set of all markings reachable from $M$. As in low level Petri nets, two enabled transitions may fire at the same time as long as they are not in conflict. An execution sequence of $N$ is $M_0[T_0/A_0 > \ldots M_k[T_k/A_k > \ldots$that is either finite when the last marking is terminal (no more enabled transition in the last marking) or infinite, each $T_i$ is an execution step consisting of a set of non-conflict firing transitions. The behavior of $N$ is the set of all execution sequences starting from the initial marking. The above semantics is step semantics, which is a generalization of interleaving semantics, but is incomparable to process semantics [19] when the net is unsafe (where a place can hold more than one token).

## III. Maude

Maude is a language and system based on rewriting logic (RWL) [20]. The following sections provide a brief introduction of the theoretical foundation and some concrete constructs of Maude.

### A. Membership Equational Logic Specification

The underlying equational logic of Maude rewriting logic is the membership equational logic (MEL) [21], which is a many-sorted logic with subsorts and overloaded function symbols. Types in MEL are called kinds, and the sorts for each kind are viewed as unary predicates. Atomic sentences are equalities $M = N$ on terms $M$ and $N$ of the same kind, and memberships $M : s$ on term $M$ and sort $s$ of the same kind. Sentences of MEL are universally quantified Horn clauses. The algebraic semantics of MEL is defined in terms of a membership equational logic specification (MES). Given a MES S, its operational semantics can be efficiently executed under certain restrictions on variables as well as confluence, termination, sort-decreasingness, and regularity [21].

### B. Rewriting Rules

A rewrite specification (RWS) $R$ consists of an underlying MES $S_R$ with a distinguished data sub-specification $S_R^D$, a set of labels $L_R$, and a set of rules $R_R$ of the form $\forall X.l : M \to N \; if \; \phi_1 \wedge \cdots \wedge \phi_n$, where $l \in L_R$ and $\phi_1 \ldots \phi_n$ are $S_R$ condition over $X$, and $M, N \in T_R(X)_k$ in $S_R$ for a rewrite kind $k$. The rewrite rule means term $M$ rewrites to term $N$ modulo the equations $E_R$ in $S_R$. The operational semantics of $R$ extends the operational semantics $S_R$ by applying both the computational equations $E_R^C$ and rewrite rules $R_R$ modulo the structural equations $E_R^S$.

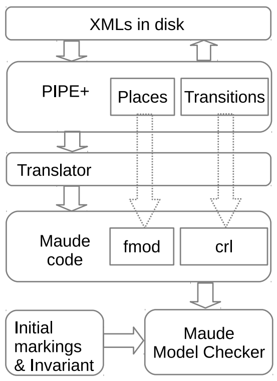

Figure 1. Overview of tool architecture

### C. Maude Language Constructs

In this section, we briefly discuss several Maude language constructs used in our translation. A Maude rewriting logic consists of one system module and several functional modules. Each functional module defines data types and the associated operations upon them through membership equational logic. The syntactic form of functional modules is as follows: $fmod\ <ModuleName>\ \ is\ \ <DeclarationsAndStatements>$ $endfm$. The mathematical semantics of a functional module defined by its initial algebra coincides with its operational semantics in terms of canonical term algebra by treating the equations as simplification rewriting rules. $<Declarations>$ can contain (1) predefined data types and their associated operations as well as already defined functional modules through the $<including>$ keyword, (2) new data types and derived data types through $<sort>$ and $<subsort>$ keywords and their plural forms respectively , (3) typed variables through keyword $<var>$ or $<vars>$, and (4) new operations through keyword $<op>$ or $<ops>$. $<Statements>$ are equations relating various defined operations using the keyword $<eq>$. The system module has the form $mod\ <ModuleName>\ is\ <Declarations, Statements, andRules>\ endm$. Rules define the state transitions in terms of conditional rewriting of the form $crl[TransitionName]\ CurrentState => NextState\ if\ Condition$.

## IV. Translating HLPNs to Maude Rewriting Systems

A theoretical framework for defining Petri nets using rewriting logic was presented in [22] and [23], in which a variety of Petri nets including several types of HLPNs were defined algebraically using the category theoretical approach [24] and operationally using rewriting logic. In this paper, we only focus on the operational aspect of translating PrT nets into a rewriting logic. Given a PrT net $PN = (N, Spec, ins)$ with a net structure $N = (P, T, F)$ and net inscription $ins = (\varphi, L, R, M_0)$, the following general translation mappings are obtained: $Spec$ is naturally translated into an underlying MES $S_{Spec}$ with a distinguished multiset sort marking; each place $p \in P$ is translated into an operation with the signature $\varphi(p) \rightarrow marking$; each transition $t \in T$ is translated into a conditional rewriting rule of the form: $t : (\bar{L}(p_1, t) \cdots \bar{L}(p_n, t)m) \rightarrow \bar{L}(t, p_1) \cdots \bar{L}(t, p_n)m)$ if $R(t)$, where $m$ is variable for marking.

### A. Translation of Places

The basic idea to translate places of PrT nets is using functional module *fmod* in Maude. The translation of places is different with regard to their types. For each place $p$ of non power set type $\varphi(p) = s1 \times ... \times sn$, we introduce a new sort $sortp$ preserving the user defined type with an unique operator $op\, OP - p :\ s1 \ldots sn \rightarrow p[ctor]$ as the constructor of tokens. For each place with power set type, in addition to define a new sort preserving the base user defined type, we define a couple of new sorts to support necessary set operations (empty, union, and difference) and to facilitate the power set manipulation.

### B. Translation of Transitions

A conditional rewriting rule is defined for each transition, with the left hand side of the rule being the incoming places, the right hand side of the rule being the outgoing places and an if statement defining the precondition (consumed tokens) and the postcondition (produced tokens). Each place associated with the transition is defined as a term, and each precondition / postcondition is a conjunct in the if statement, where = is used for a comparison (reading) and := is used for assignment (writing). Currently, we impose a few restrictions on the transition constraint formulas such that only existential quantifiers are permitted since Maude does not support quantifiers. The existential quantifiers in a constraint formula can be effectively handled by replacing an arc set variable with set element variables in braces, for example, a transition constraint of the form $\exists x \in V.(\exists y \in V.(F(x, y)))$, where $V$ is an arc set variable, becomes $F(x, y)$ with $V$ being replaced by arc set element variables $\{x, y\}$. This restriction may affect the analysis of some models that require the use of universal quantifiers; however it does not cause problems for dealing with many real world applications. Another minor restriction requires that the input and output arc variables be distinct.

### C. Defining Model Checking Module

Maude provides a simple search mechanism for proving an invariant through finding a counter example of its negation in the following form: search init =>* M:Marking such that not $Inv(M : Marking)$. $Inv$ is an invariant condition defined using Maude equations. To check the generated Maude model, we define a module including the initial marking equations as well as the negated invariant equations.

### D. The Translator

The translation method has been implemented in our PIPE+ environment [15] and is completely automatic except for the initial marking and the property translation. The translator and its environment is shown in Figure 1.

## V. Experiment Results

We conducted the experiments with a machine that has 2.3Ghz quad-core CPU and 16GB memory. We verified the invariant property of the Mondex [4] using Maude. To examine the performance and scalability of our method, we conducted a few more experiments on several well known

Table I
EXPERIMENT RESULTS FOR THE READER WRITER MODEL

| Reader | Writer | Uses (MB) | Seconds | States | Rewrites |
|---|---|---|---|---|---|
| 3 | 3 | 2 | 0.002 | 77 | 507 |
| 4 | 4 | 2 | 0.005 | 180 | 1177 |
| 5 | 5 | 2.2 | 0.012 | 407 | 2639 |
| 10 | 10 | 11 | 0.7 | 21714 | 138877 |
| 12 | 12 | 42 | 3.2 | 102700 | 657737 |
| 15 | 15 | 438 | 68 | 1016273 | 6524523 |
| 16 | 16 | 915 | 248 | 2163216 | 13897825 |

examples selected from the benchmark problems of Model Checking Contest @ Petri Nets [10], which is held annually to assess Petri nets based formal verification tools and techniques. These examples include dinning philosophers problem, reader writer problem, and shared memory problem for which there were high level Petri net models and all the tools failed model checking during the contest. The example HLPNs and translated Maude code can be found at https://github.com/liusu1011/PIPE-Verifier/tree/reng/panther/testcases/2016. To provide a glimpse of the experiment results, the running results of the reader writer system are shown in Table I; where the high level Petri net model has 11 places and 10 transitions. It can be seen that both the space and time increase exponentially with the increasing numbers of readers and writers.

## VI. CONCLUDING REMARKS

In this paper, we presented an approach to translate a particular type of high level Petri nets (HLPNs), predicate transition nets (PrT nets), into a Maude specification and then leverage the Maude model checking capability to analyze invariant properties in the given PrT nets. The supporting tool automatically generates a Maude specification from a given PrT net model. Although the close relationship between Petri nets and term rewriting systems was studied and a general translation framework was established a decade ago [22], we believe that our method is the first concrete realization of the general framework. This translation and analysis method based on Maude complements and enriches our existing analysis methods and tool environment PIPE+ for analyzing PrT nets using simulation, model checking based on SPIN, and bounded model checking based on Z3. We have added real data type into PIPE+ recently that will enable our translation method to model and analyze cyber physical systems. We are currently developing a general first order linear time temporal logic (FOLTL) editor and building a FOLTL formula translator to integrate into our environment. This will allow us to explore the full capability of Maude model checking. Furthermore, we will investigate the integration of Maude and an SMT solver as our analysis engine that can support a full fledge translation method to cover quantifiers in transition constraints.

**Acknowledgments** Xudong He was partially supported by the NSF under grant HRD-0833093 and by the AFRL under agreement number FA8750-15-2-0106. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government. We thank the anonymous reviewers for their helpful comments.

## REFERENCES

[1] V. Khomenko and M. Koutny, "Branching Processes of High-Level Petri Nets," in *Tools and Algorithms for the Construction and Analysis of Systems* (H. Garavel and J. Hatcliff, eds.), no. 2619 in Lecture Notes in Computer Science, pp. 458–472, Springer Berlin Heidelberg, Apr. 2003.
[2] X. He, H. Yu, T. Shi, J. Ding, and Y. Deng, "Formally analyzing software architectural specifications using SAM," *J. Syst. Softw.*, vol. 71, no. 1-2, pp. 11–29, 2004.
[3] G. Argote-Garcia, P. J. Clarke, X. He, Y. Fu, and L. Shi, "A Formal Approach for Translating a SAM Architecture to PROMELA," in *SEKE*, pp. 440–447, 2008.
[4] R. Zeng and X. He, "Analyzing a Formal Specification of Mondex Using Model Checking," *Theoretical Aspects of Computing–ICTAC 2010*, pp. 214–229, 2010.
[5] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
[6] "CPN Tools Homepage." http://cpntools.org/.
[7] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2*. London, UK, UK: Springer-Verlag, 1995.
[8] S. Hostettler, A. Marechal, A. Linard, M. Risoldi, and D. Buchs, "High-Level Petri Net Model Checking with AlPiNA," *Fundam. Inf.*, vol. 113, pp. 229–264, Aug. 2011.
[9] K. L. McMillan, "Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits," in *Proceedings of the Fourth International Workshop on Computer Aided Verification*, CAV '92, (London, UK, UK), pp. 164–177, Springer-Verlag, 1993.
[10] "Model checking contest @ Petri Nets 2013," (Milano, Italy), June 2013.
[11] S. Katz and O. Grumberg, "A Framework for Translating Models and Specifications," in *Proceedings of the Third International Conference on Integrated Formal Methods*, pp. 145–164, Springer-Verlag, 2002.
[12] "The SMV System." {http://www.cs.cmu.edu/~modelcheck/smv.html}.
[13] G. Holzmann, *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.
[14] S. Liu, R. Zeng, and X. He, "PIPE+ - A Modeling Tool for High Level Petri Nets," in *Proc. of International Conference on Software Engineering and Knowledge Engineering*, (Miami), pp. 115–121, 2011.
[15] S. Liu, R. Zeng, Z. Sun, and X. He, "SAMAT - A Tool for Software Architecture Modeling and Analysis," in *Proceedings of the 24th International Conference on Software Engineering {\&} Knowledge Engineering (SEKE'2012)*, (San Francisco Bay, USA), pp. 352–358, 2012.
[16] X. He, "A Formal Definition of Hierarchical Predicate Transition Nets," in *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, (London, UK, UK), pp. 212–229, Springer-Verlag, 1996.
[17] W. Reisig, "Petri nets and algebraic specifications," *Theoretical Computer Science*, vol. 80, pp. 1–34, Mar. 1991.
[18] H. J. Genrich and K. Lautenbach, "System modelling with high-level Petri nets," *Theoretical Computer Science*, vol. 13, pp. 109–135, Jan. 1981.
[19] E. Best and R. Devillers, "Sequential and Concurrent Behaviour in Petri Net Theory," *Theor. Comput. Sci.*, vol. 55, pp. 87–136, Nov. 1987.
[20] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *All About Maude - a High-performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic*. Berlin, Heidelberg: Springer-Verlag, 2007.
[21] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer, "Specification and Proof in Membership Equational Logic," *Theor. Comput. Sci.*, vol. 236, pp. 35–132, Apr. 2000.
[22] M.-O. Stehr, J. Meseguer, and P. C. Ölveczky, "Rewriting Logic As a Unifying Framework for Petri Nets," in *Unifying Petri Nets, Advances in Petri Nets*, (London, UK, UK), pp. 250–303, Springer-Verlag, 2001.
[23] M.-O. Stehr, J. Meseguer, and P. C. Ölveczky, "Representation and Execution of Petri Nets Using Rewriting Logic as a Unifying Framework," *Electronic Notes in Theoretical Computer Science*, vol. 44, pp. 140–162, July 2001.
[24] J. Meseguer and U. Montanari, "Petri nets are monoids," *Information and Computation*, vol. 88, pp. 105–155, Oct. 1990.