PReMo:
An Analyzer for Probabilistic Recursive Models

Dominik Wojtczak and Kousha Etessami

School of Informatics, University of Edinburgh

Abstract. This paper describes PReMo, a tool for analyzing Recursive Markov Chains, and their controlled/game extensions: (1-exit) Recursive Markov Decision Processes and Recursive Simple Stochastic Games.

1 Introduction

Recursive Markov Chains (RMCs) [4,5] are a natural abstract model of probabilistic procedural programs and other systems involving recursion and probability. They are formally equivalent to probabilistic Pushdown Systems (pPDSs) ([2,3]), and they define a class of infinite-state Markov chains that generalize a number of well studied stochastic models such as Stochastic Context-Free Grammars (SCFGs) and Multi-Type Branching Processes. In a series of recent papers ([4,5,6,7]), the second author and M. Yannakakis have developed algorithms for analysis and model checking of RMCs and their controlled and game extensions: 1-exit Recursive Markov Decision Processes (1-RMDPs) and 1-exit Recursive Simple Stochastic Games (1-RSSGs). These extensions allow modelling of nondeterministic and interactive behavior.

In this paper we describe PReMo, a software tool for analysing models based on RMCs, 1-RMDPs, and 1-RSSGs. PReMo allows these models to be specified in several different input formats, including a simple imperative-style language for specifying RMCs and RSSGs, and an input format for SCFGs. For RMCs/RSSGs, PReMo generates a graphical depiction of the model, useful for visualizing small models (see Figure 1). PReMo has implementations of numerical algorithms for a number of analyses of RMCs and 1-RSSGs. From an RMC, PReMo generates a corresponding system of nonlinear polynomial equations, whose Least Fixed Point (LFP) solution gives precisely the termination probabilities for vertex-exit pairs in the RMC. For 1-RSSGs, it generates a system of nonlinear min-max equations, whose LFP gives the *values* of the termination game starting at each vertex. Computation of termination probabilities is a key ingredient for model checking and other analyses for RMCs and pPDSs ([4,5,2]). PReMo provides a number of optimized numerical algorithms for computing termination probabilities. Methods provided include both dense and sparse versions of a decomposed Newton's method developed in [4], as well as versions of value iteration, optimized using nonlinear generalizations of Gauss-Seidel and SOR techniques. The latter methods also apply to analysis of 1-RSSGs.

In addition to computing termination probabilities, PReMo can compute the (maximum/minimum/game) *expected termination time* in 1-RMCs, 1-RMDPs,
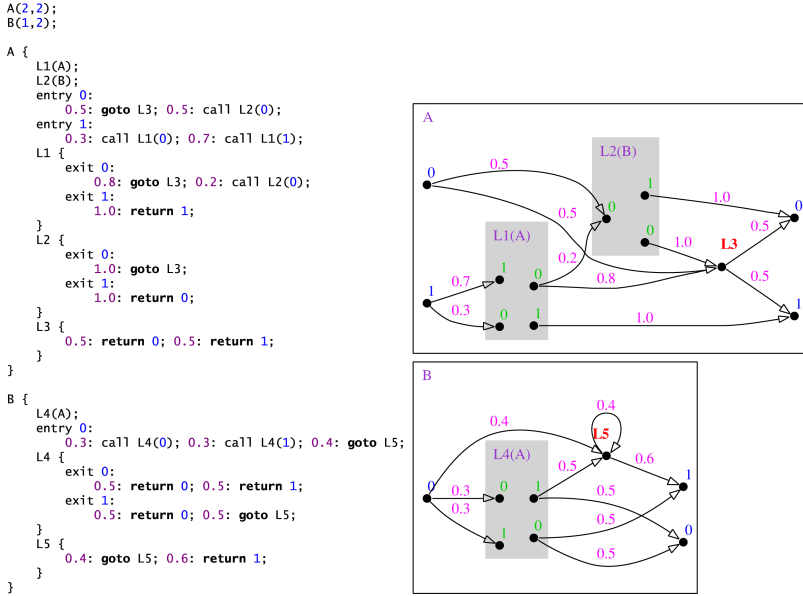
```
A(2,2);
B(1,2);

A {
    L1(A);
    L2(B);
    entry 0:
        0.5: goto L3; 0.5: call L2(0);
    entry 1:
        0.3: call L1(0); 0.7: call L1(1);
    L1 {
        exit 0:
            0.8: goto L3; 0.2: call L2(0);
        exit 1:
            1.0: return 1;
    }
    L2 {
        exit 0:
            1.0: goto L3;
        exit 1:
            1.0: return 0;
    }
    L3 {
        0.5: return 0; 0.5: return 1;
    }
}

B {
    L4(A);
    entry 0:
        0.3: call L4(0); 0.3: call L4(1); 0.4: goto L5;
    L4 {
        exit 0:
            0.5: return 0; 0.5: return 1;
        exit 1:
            0.5: return 0; 0.5: goto L5;
    }
    L5 {
        0.4: goto L5; 0.6: return 1;
    }
}
```


Fig. 1. Source code of an RMC, and its visualization generated by PReMo

and 1-RSSGs. It does so by generating a different monotone system of linear (min-max) equations, whose LFP is the value of the game where the objectives of the two players are to maximize/minimize the expected termination time (these expected times can be infinity). (This analysis extends, to a game setting, the expected reward analysis for pPDSs (equivalently, RMCs) studied in [3]. The generalization works for 1-RMDPs and 1-RSSGs, which correspond to controlled/game versions of *stateless* pPDSs, also known as pBPAs. We do not explicate the theory behind these game analyses here. It is a modification of results in [6,7], and will be explicated elsewhere.)

PReMo is implemented entirely in Java, and has the following main components: (1) A parsers for text descriptions of RMCs, RSSGs, and SCFGs, using one of several input formats; (2) A menu-driven GUI (using the Standard Widget Library(SWT)), with an editor for different input formats, and menu choices for running different analyses with different methods; (3) A graphical depiction generator for RMCs and RSSGs, which produces output using the `dot` format. (4) *Optimized solvers:* Several solvers are implemented for computation of termination probabilities/values for RMCs and 1-RSSGs, and also computation of expected termination times for 1-RMCs, 1-RMDPs, 1-RSSGs. We conducted a range of experiments. Our experiments indicate very promising potential for several methods. In particular, our decomposed Sparse Newton's method performed very well on most models we tried, up to quite large sizes. Although these numerical methods appear to work well in practice on most instances, there are no

theoretical guarantees on their performance, and there are many open questions about the complexity of the underlying computational problems (see [4,5,6,7]).

We can see PReMo source code for an RMC, together with a visualization that PReMo generates for it, in Figure 1. Informally, an RMC consists of several component Markov Chains (in Fig. 1, these are named A and B) that can call each other recursively. Each component consists of nodes and boxes with possible probabilistic transitions between them. Each box is mapped to a specific component so that every time we reach an entry of this box, we jump to the corresponding entry of the component it is mapped to. When/if we finally reach an exit node of that component, we will jump back to a respective exit of the box that we have entered this component from. This process models, in an obvious way, function invocation in a probabilistic procedural program. Every potential function call is represented by a box. Entry nodes represent parameter values passed to the function, while exit nodes represent returned values. Nodes within a component represent control states inside the function. Documentation about the input languages is available on the PReMo web page.

The core numerical computation for all the analyses provided by PReMo involves solving a monotone systems of nonlinear min-max equations. Namely, we have a vector of variables $\mathbf{x} = (x_1, \ldots, x_n)$, and one equation per variable of the form $x_i = P_i(\mathbf{x})$, where $P_i(\mathbf{x})$ is a polynomial-min-max expression with rational coefficients. In vector notation, this system of equations can be denoted $\mathbf{x} = P(\mathbf{x})$. The goal is to find the Least Fixed Point solution, i.e., the least non-negative solution, $\mathbf{q}^* \in \mathbb{R}^n_{\geq 0}$, of these equations, which is $\lim_{k \to \infty} P^k(\mathbf{0})$. In brief, the solvers in PReMo work as follows (see [4,6] for more background). First, we decompose the equations into SCCs and calculate the solution "bottom-up", solving the Bottom SCCs first and plug in the solution as constants in higher SCCs. To solve each SCC, PReMo provides several methods:

Value iteration: nonlinear Jacobi & Gauss-Seidel. Optimized forms of nonlinear *value iteration* have been implemented for computing the LFP of $\mathbf{x} = P(\mathbf{x})$. Jacobi, or basic iteration, just computes $\mathbf{x}^0 = \mathbf{0}, \mathbf{x}^1, \mathbf{x}^2, \ldots$, where $\mathbf{x}^i = P(\mathbf{x}^{i-1})$. Gauss-Seidel iteration optimizes this slightly: inductively, having computed x_j^{k+1} for $j < i$, let $x_i^{k+1} := P_i(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^k, x_{i+1}^k, \ldots, x_n^k)$. Successive Overrelaxation (SOR) is an "optimistic" modification of Gauss-Seidel, which isn't guaranteed to converge in our case.

Dense and sparse decomposed Newton's method. Newton's method attempts to compute solutions to $F(\mathbf{x}) = \mathbf{0}$. In n-dimensions, it works by iterating $\mathbf{x}^{k+1} := \mathbf{x}^k - (F'(\mathbf{x}^k))^{-1} F(\mathbf{x}^k)$ where $F'(\mathbf{x})$ is the *Jacobian* matrix of partial derivatives of F. In our case we apply this method for $F(\mathbf{x}) = P(\mathbf{x}) - \mathbf{x}$. It was shown in [4] that if the system is decomposed into SCCs appropriately, convergence to the LFP is guaranteed, if we start with $\mathbf{x}^0 = \mathbf{0}$. The expensive task at each step of Newton is the matrix inversion $(F'(\mathbf{x}^k))^{-1}$. Explicit matrix inversion is too expensive for huge matrices. But this matrix is typically sparse for RMCs, and we can handle much larger matrices if instead of inverting $(F'(\mathbf{x}^k))$ we solve the following equivalent sparse linear system of equations: $(F'(\mathbf{x}^k))(\mathbf{x}^{k+1} - \mathbf{x}^k) = F(\mathbf{x}^k)$ to compute the value of $\mathbf{x}^{k+1} - \mathbf{x}^k$, and then add $\mathbf{x}^k$ to obtain $\mathbf{x}^{k+1}$. We used

the solver library MTJ (Matrix Toolkit for Java) and tried various sparse linear solvers. Our Dense Newton's method uses LU decomposition to invert $(F'(\mathbf{x}^k))$.

Iterative numerical solvers can only converge to within some error to the actual solution. PReMo provides different mechanisms for users to choose when to stop the iteration: absolute tolerance, relative tolerance, and a specified number of iterations. In, e.g., the absolute tolerance mode, the algorithm stops after the first iteration when the absolute difference in the value for all variables changed less than a given $\epsilon > 0$. This does not in general guarantee closeness to the actual solution, but it behaves well in practice.

2   Experimental Results

We ran a wide range of experiments on a Pentium 4 3GHz with 1GB RAM, running Linux Fedora 5, kernel 2.6.17, using Java 5.0. Please see our fuller report [9] for more details about our experimental results.

SCFGs generated from the Penn Treebank NLP corpora. We checked the *consistency*[1] of a set of large SCFGs, with 10,000 to 50,000 productions, used by the Natural Language Processing (NLP) group at University of Edinburgh and derived by them from the Penn Treebank NLP corpora. These SCFGs were assumed to be consistent by construction. Our most efficient method (Sparse Newton) solved all these SCFGs in a few seconds (see Table 1). Two out of seven SCFGs were (very) inconsistent, namely those derived from the `brown` and `switchboard` corpora of Penn Treebank, with termination probabilities as low as 0.3 for many nonterminals. This inconsistency was a surprise to our NLP colleagues, and was subsequently identified by them to be caused by annotation errors in Penn Treebank itself ([1]). Note that both dense and sparse versions of decomposed Newton's method are by far the fastest. Since the largest SCCs are no bigger than 1000 vertices, dense Newton also worked on these examples. Most of the time for Newton's method was in fact taken up by the initialization phase, for computing all the partial derivatives in entries of the Jacobian $F'(\mathbf{x})$. We thus optimized the computation of the Jacobian in several ways.

Randomly generated RMCs and 1-RSSGs. We tested PReMo on randomly generated RMCs of different sizes, ranging from 10,000 to 500,000 nodes (variables). In random large instances, with very high probability most nodes are in one huge SCC with small diameter ("small world phenomenon"). Dense Newton's method did not work at all on these huge SCCs, because inverting such large matrices is too costly, but both Gauss-Seidel and Sparse Newton did very well. In particular, Sparse Newton handled instances with 500,000 variables in ~ 45 seconds. For random 1-RSSGs, although we have no Newton's method available for 1-RSSGs, value iteration performed well (see [9]).

Quicksort. For expected termination time analyses, we considered a toy model of randomized Quicksort, using a simple hierarchical 1-RMC. The model has

---

[1] An SCFG is called *consistent* if starting at all nonterminals in the grammar, a random derivation terminates, and generates a finite string, with probability 1.

Table 1. Performance results for checking consistency of SCFGs derived from Penn Treebank. Time is in seconds. In parentheses is the number of iterations for the biggest SCC. Stopping condition: absolute tolerance $\epsilon = 10^{-12}$. SCFG was declared "consistent" if all nonterminals had termination probability $\geq (1 - 10^{-4})$. The SCFGs `brown` and `swbd` failed consistency by a wide margin.

name	#prod	max-scc	Jacobi	Gauss Seidel	SOR ω=1.05	DNewton	SNewton
brown	22866 ✗	448	312.084(9277)	275.624(7866)	diverge	2.106(8)	2.115(9)
lemonde	32885 ✓	527	234.715(5995)	30.420(767)	diverge	1.556(7)	2.037(7)
negra	29297 ✓	518	16.995(610)	4.724(174)	4.201(152)	1.017(6)	0.499(6)
swbd	47578 ✗	1123	445.120(4778)	19.321(202)	25.654(270)	6.435(6)	3.978(6)
tiger	52184 ✓	1173	99.286(1347)	16.073(210)	12.447(166)	5.274(6)	1.871(6)
tuebadz	8932 ✓	293	6.894(465)	1.925(133)	6.878(461)	0.477(7)	0.341(7)
wsj	31170 ✓	765	462.378(9787)	68.650(1439)	diverge	2.363(7)	3.616(8)

n components, Q_i, $i = 1, \ldots, n$, corresponding to invocations of Quicksort on arrays of size i. Component Q_i takes time i to pivot and split the entries, and then recurses on the two partitions. This is modeled by transitions of probability $1/(i-1)$, for each $d \in \{1, \ldots, i-1\}$, to two sequential boxes labeled by Q_d and Q_{i-d}. We computed expected termination time for various sizes n. We also tried letting the pivot be controlled by the *minimizer* or *maximizer*, and we computed optimal expected running time for such 1-RMDPs, in order to consider best-case and worst-case running times of Quicksort. As expected, the results fitted the well-known theoretical analysis of $\Theta(n \log n)$ and $\Theta(n^2)$ for running times of randomized/best-case, and worst-case Quicksort, respectively.

3   Future Work

The next important step is to extend the RMC language to allow variables and conditional branching, i.e., probabilistic Boolean Programs. We are working toward implementation of a full-fledged linear-time model checker for RMCs. This is a major challenge because there are very difficult numerical issues that have to be overcome in order to enable general model checking. PReMo 1.0 is available at: `http://homepages.inf.ed.ac.uk/s0571094/PReMo`

References

1. A. Dubey and F. Keller. *personal communication*, 2006.
2. J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *Proc. LICS'04*, 2004.

3. J. Esparza, A. Kučera, and R. Mayr. Quantitative Analysis of Probabilistic Pushdown Automata: Expectations and Variances. In *Proc. LICS'05*, 2005.
4. K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. In *Proc. STACS'05*, 2005.
5. K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic state machines. In *Proc. TACAS'05*, 2005.
6. K. Etessami and M. Yannakakis. Recursive markov decision processes and recursive stochastic games. In *Proc. ICALP'05*, 2005.
7. K. Etessami and M. Yannakakis. Efficient qualitative analysis of classes of recursive markov decision processes and simple stochastic games. In *Proc. STACS'06*, 2006.
8. M. J. Neiderhof and G. Satta. Using Newton's method to compute the partition function of a PCFG, 2006. *unpublished draft manuscript*.
9. D. Wojtczak and K. Etessami. PReMo: an analyzer for Probabilistic Recursive Models. Fuller report, with more experimental data. `http://homepages.inf.ed.ac.uk/s0571094/PReMo/tacas07premo-long.pdf`