PReMo: An analyzer for Probabilistic Recursive Models

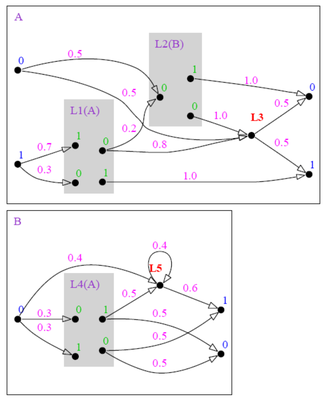
Dominik Wojtczak and Kousha Etessami

PReMo is a tool for analyzing Recursive Markov Chains(RMCs), and their controlled/game extensions. RMCs are a natural abstract model of probabilistic procedural programs and other systems involving recursion and probability. It generalizes a number of well studied stochastic models such as Stochastic Context-Free Grammars (SCFGs) and Multi-Type Branching Processes.

What is a Recursive Markov Chain?

We can see below a source code of an example RMC, together with a visualization that PReMo generates for it:

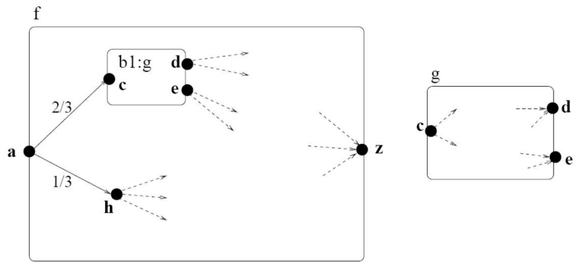
```
A(2,2);
B(1,2);

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;
    }
}
```



Informally, an RMC consists of several component Markov Chains (in this example 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.

What PReMo can do and how?

PReMo can compute the probability of termination at a given exit for an RMC starting at any of the vertices. It does so by generating and finding the Least Fixed Point(LFP) of the underlying system of a non-linear (min-max) equations. We can see how this system is constructed in the following example (where e.g. $x_{(f,a,z)}$ denotes the probability of termination at the exit z of the component f starting at the node a, for details see [**?**]):



- exit z: $x_{(f,z,z)} = 1$
- node a: $x_{(f,a,z)} = \frac{1}{3}x_{(f,h,z)} + \frac{2}{3}x_{(f,(b1,c),z)}$
- box b1 entry c: $x_{(f,(b1,c),z)} = x_{(g,c,d)}x_{(f,(b1,d),z)} + x_{(g,c,e)}x_{(f,(b1,e),z)}$
- do it for every single node and get a system of equations $\mathbf{x} = \mathbf{P}(\mathbf{x})$

PReMo can also compute the (optimal/pessimal) *expected termination time* in a restricted subclass of RMCs (and their game extensions) that are allowed to have just one exit. It does so by generating a different monotone system of linear (min-max) equations (see [**?**] for details) (notice that these expected times can be infinite).

In brief, the solvers in PReMo work as follows:

1. decompose the equation system $\mathbf{x} = \mathbf{P}(\mathbf{x})$ into Strongly Connected Components(SCCs)
2. for each SCC S (starting at the bottom ones):
   (a) run some numerical method to find LFP of $\mathbf{P}|_S(\mathbf{x}|_S)$
   (b) plug-in the approximate solution for the variables in S into higher SCCs

To solve each SCC, PReMo provides several methods:

- *Jacobi*(or basic iteration) starts with $\mathbf{x}^0 = \mathbf{0}$ and iterates $\mathbf{x}^i = P(\mathbf{x}^{i-1})$
- *Gauss-Seidel* improves on *Jacobi*, by using the value of variables from the current step, not the previous one, if they are already computed
- *Successive Overrelaxation (SOR)* is an "optimistic" modification of Gauss-Seidel, which isn't guaranteed to converge in our case
- *n-dimensional Newton's method* computes solutions to $F(\mathbf{x}) = \mathbf{0}$, 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 it to $F(\mathbf{x}) = P(\mathbf{x}) - \mathbf{x}$.
  - *Dense Newton(DNewton)* uses LU decomposition to invert $(F'(\mathbf{x}^k))$
  - *Sparse Newton(SNewton)*, rather than inverting $F'(\mathbf{x}^k)$, it solves a sparse linear system $F'(\mathbf{x}^k)(\mathbf{x}^{k+1} - \mathbf{x}^k) = F(\mathbf{x}^k)$ in order to compute the value of $\mathbf{x}^{k+1} - \mathbf{x}^k$, and then by adding $\mathbf{x}^k$, it obtains $\mathbf{x}^{k+1}$

Experimental results

All experiments were run on a Pentium 4 3GHz with 1GB RAM.

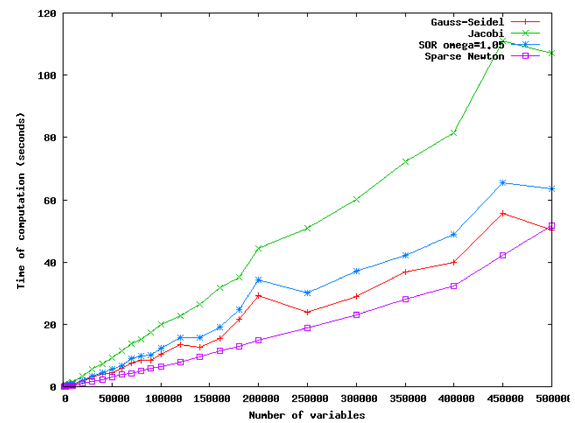
- SCFGs derived from the Penn Treebank

  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. We checked the *consistency* of a large SCFGs used by the Natural Language Processing (NLP) group at University of Edinburgh and derived by them from the Penn Treebank NLP corpora. Two out of seven grammars turned out to be very inconsistent, namely those derived from the `brown` and `switchboard` corpora of Penn Treebank. This inconsistency was subsequently identified to be caused by annotation errors in the Penn Treebank itself.

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)

number of productions, the biggest SCC's size, running time in seconds and in parentheses the maximum number of iterations

- Randomly generated RMCs of various sizes



As we can see Sparse Newton is better than any other iterative method.

References

[1] PReMo's homepage http://groups.inf.ed.ac.uk/premo

[2] K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of non-linear equations. In *Proc. of 22nd STACS'05*. Springer, 2005.

[3] K. Etessami, D. Wojtczak, and M. Yannakakis Recursive Stochastic Games with Positive Rewards. *Submitted for publication.*