SCOOP: A Tool for SymboliC Optimisations Of Probabilistic Processes

Mark Timmer
Formal Methods and Tools Group
University of Twente, The Netherlands
Email: timmer@cs.utwente.nl

***Abstract*—This paper presents SCOOP: a tool that symbolically optimises process-algebraic specifications of probabilistic processes. It takes specifications in the prCRL language (combining data and probabilities), which are linearised first to an intermediate format: the LPPE. On this format, optimisations such as dead-variable reduction and confluence reduction are applied automatically by SCOOP. That way, drastic state space reductions are achieved while never having to generate the complete state space, as data variables are unfolded only locally. The optimised state spaces are ready to be analysed by for instance CADP or PRISM.**

I. Introduction

Several algorithms and tools exist for model checking qualitative and quantitative properties for a wide range of probabilistic models, modelling for instance randomised protocols or biological processes. Although these techniques are promising, their applicability is limited by the well-known *state-space explosion* and the restricted treatment of *data*.

Probabilistic process algebras typically only allow a random choice over a fixed distribution, and input languages for probabilistic model checkers such as the PRISM language [5] or the probabilistic variant of Promela [1] only support basic data types. However, to model realistic systems, more elaborate and convenient means for data modelling are indispensable.

The incorporation of data yields a significant increase of state space size. However, most of today's probabilistic minimisation techniques are not well-suited to be applied in the presence of data. Although several reduction techniques can be applied at the model level, this requires the complete model to be constructed first. Our approach is to *symbolically* reduce at the process-algebraic level, minimising state spaces prior to their generation by means of syntactic transformations.

We developed the probabilistic process-algebraic language prCRL [6], [7], generalising the μCRL language [4] by adding a probabilistic choice operator. We also defined a restricted variant of prCRL, called the LPPE (linear probabilistic process equation). Any prCRL specification fulfilling some mild conditions can be transformed (linearised) into an LPPE, which then allows symbolic reductions and easy state space generation (in the form of a probabilistic automaton). On the LPPE format, reductions can be applied. Most of these transform an LPPE into an equivalent LPPE, others work during state space generation. Figure 1 illustrates the approach.

The SCOOP tool we present takes a prCRL specification, linearises it, applies reduction techniques, and generates the (often significantly) reduced state space, without having to construct the complete state space first. The tool is publicly available, open source and has a user-friendly web interface[1].

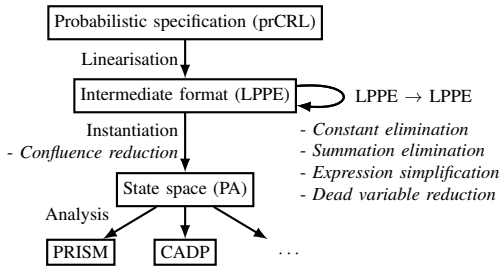

Fig. 1. The LPPE-based verification approach.

II. The input language

The input language for SCOOP is the prCRL language, introduced in [6], [7]. The core of the language consists of process terms p, conforming to the following grammar:

$$p ::= Y(\boldsymbol{t}) \;\mid\; c \Rightarrow p \;\mid\; p + p \;\mid\; \sum_{\boldsymbol{x}:\boldsymbol{D}} p \;\mid\; a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p$$

Here, Y is a process name, $\boldsymbol{t}$ a vector of expressions, c a boolean expression, $\boldsymbol{x}$ a vector of variables ranging over countable type $\boldsymbol{D}$, a a (parameterised) atomic action, and f a real-valued expression yielding values in $[0, 1]$. A system specification then consists of a set of processes (defined by such process terms), an initial state and a data specification. As an illustration of the language, observe the following example.

$$X = \sum_{n:\{1,2,3\}} \text{write}(n) \sum_{i:\{1,2\}} \tfrac{i}{3} : (i = 1 \Rightarrow X + i = 2 \Rightarrow \text{beep} \cdot X)$$

This system continuously chooses and writes a number 1, 2 or 3 in a nondeterministic manner. Between each pair of write actions, it beeps with probability $2/3$.

The language also supports parallel composition of processes, as well as action renaming, encapsulation and hiding.

The LPPE format is a restriction of the prCRL language, only allowing a single process definition in which every probabilistic choice is immediately followed by a process instantiation. For more details on prCRL and the LPPE format, we refer to [6], [7]. There, we also show how every reasonable prCRL specification can be linearised to an equivalent LPPE.

[1]The implementation, including the web-based interface, can be found at http://www.ewi.utwente.nl/~timmer/scoop/.

III. Optimisation techniques

Using the LPPE, several reduction techniques can be applied. We distinguish between (1) *LPPE simplification techniques* and (2) *state space reduction techniques*.

SCOOP implements three LPPE simplification techniques, which do not change the actual state space, but improve readability and speed up state space generation. (1) *Constant elimination* detects if a parameter of an LPPE never changes its value. Then, the parameter is omitted and every reference to it is replaced by its initial value. (2) *Summation elimination* simplifies nondeterministic choices for which only one of the possible alternatives enables real behaviour. This kind of construction often occurs when composing parallel components that communicate using message passing. (3) *Expression simplification* rewrites conditions, action parameters and next state parameters using for instance basic logical identities and the evaluation of functions, where possible using heuristics.

SCOOP implements two state space reduction techniques, that do change the LPPE or instantiation in such a way that the resulting state space will be smaller. (1) *Dead variable reduction* was generalised easily from the non-probabilistic variant, introduced in [9]. It reduces state spaces while preserving strong bisimulation, by resetting irrelevant variables based on the control flow of an L(P)PE. (2) *Confluence reduction* was introduced in [2] for LPEs, to reduce state spaces while preserving branching bisimulation. Basically, it detects internal τ-transitions that do not influence a system's behaviour, and uses this information when generating the state space. This reduces the number of states that have to be visited during state space generation, and even more the number of states that actually have to be stored. A generalisation to the probabilistic setting was presented in [8].

All these techniques work on the syntactic level, i.e., they do not unfold the data types at all, or only locally to avoid a data explosion. Hence, a smaller state space is obtained without first having to generate the original one.

IV. The SCOOP tool

SCOOP was developed in Haskell (6640 lines of code excluding comments), based on a simple data language to allow the modelling of several kinds of protocols and systems. A web-based interface makes it convenient to use; it provides the user with 30 seconds of server-time per request. Alternatively, SCOOP can be downloaded to run locally on any platform.

The tool automatically linearises prCRL specifications while applying the LPPE simplification techniques, and allows the user to choose whether or not to apply dead variable reduction and/or confluence reduction.

After generating and optimising an LPPE, SCOOP can also generate its state space and display it in several ways. It can export to the AUT format for analysis with the CADP toolset [3], or to a transition matrix for analysis using PRISM [5]. Alternatively, under some conditions SCOOP can translate the optimised LPPE directly to the PRISM language, allowing the use of this probabilistic model checker to symbolically compute (quantitative) properties of the model.

TABLE I
State space reduction using SCOOP.

Spec.	Original States	Reduced States	Visited States	Running time (sec) Before	After
`1-3-15`	1,043,635	68,926	251,226	313.35	65.96
`1-3-18`	2,028,181	118,675	428,940	1161.58	124.74
`1-3-21`	out of mem.	187,972	675,225	–	205.90
`1-3-27`	out of mem.	398,170	1,418,220	–	497.94
`1-4-5`	759,952	61,920	300,569	322.62	75.14
`1-4-6`	1,648,975	127,579	608,799	1073.16	155.74
`1-4-7`	out of mem.	235,310	1,108,391	–	291.25
`1-4-8`	out of mem,	400,125	1,865,627	–	1069.56
`1-5-2`	260,994	14,978	97,006	155.37	29.40
`1-5-3`	out of mem.	112,559	694,182	–	213.10

Case studies. To illustrate the strengths of SCOOP, we applied it to several variants of a leader election protocol with `i` parties each throwing a `j`-sided die (referred to in Table I by `1-i-j`. Both state space reduction techniques were applied, decreasing the number of states by 90% – 95%. The number of transitions (not shown here) decreased even more.

The time needed to generate the reduced state spaces was always at most one fourth of the time to generate the original ones (up until the point where swapping was needed; then, the generation of the smaller state spaces is even faster, relatively).

Acknowledgment

This research has been partially funded by NWO under grant 612.063.817 (SYRUP) and grant Dn 63-257 (ROCKS), and by the EU under FP7-ICT-2007-1 grant 214755 (QUASIMODO). We thank Axel Belinfante for providing a web-based interface for the SCOOP tool using his generic framework for command-line tools (http://www.purl.org/net/puptol).

References

[1] C. Baier, F. Ciesinski, and M. Größer. PROBMELA: a modeling language for communicating probabilistic processes. In *Proc. of the 2nd ACM/IEEE Int. Conf. on Formal Methods and Models for Co-Design (MEMOCODE)*, pages 57–66. IEEE, 2004.

[2] S.C.C. Blom and J.C. van de Pol. State space reduction by proving confluence. In *Proc. of the 14th Int. Conf. on Computer Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 596–609. Springer, 2002.

[3] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A toolbox for the construction and analysis of distributed processes. In *Proc. of the 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6605 of *LNCS*, pages 372–387. Springer, 2011.

[4] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Proc. of Algebra of Communicating Processes*, Workshops in Computing, pages 26–62. Springer, 1995.

[5] A. Hinton, M.Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of the 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[6] J.-P. Katoen, J.C. van de Pol, M.I.A. Stoelinga, and M. Timmer. A linear process-algebraic format for probabilistic systems with data. In *Proc. of the 10th Int. Conf. on Application of Concurrency to System Design (ACSD)*, pages 213–222. IEEE Computer Society, 2010.

[7] J.-P. Katoen, J.C. van de Pol, M.I.A. Stoelinga, and M. Timmer. A linear process-algebraic format with data for probabilistic automata. *Theoretical Computer Science*, to be published, 2011.

[8] M. Timmer, M.I.A. Stoelinga, and J.C. van de Pol. Confluence reduction for probabilistic systems. In *Proc. of the 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6605 of *LNCS*, pages 311–325. Springer, 2011.

[9] J.C. van de Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. In *Proc. of the 7th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, volume 5799 of *LNCS*, pages 54–68. Springer, 2009.