SMART: the Stochastic Model checking Analyzer for Reliability and Timing*

Gianfranco Ciardo  
University of California, Riverside  
ciardo@cs.ucr.edu

Andrew S. Miner  
Iowa State University  
asminer@cs.iastate.edu

SMART provides a seamless environment for the logic and probabilistic analysis of complex systems, for use in both the classroom and industrial applications. While initially designed as a powerful stochastic environment integrating multiple modeling formalisms, SMART now includes logical analysis and employs some of the most efficient data structures and algorithms for the analysis of discrete-state systems. For logical behavior, explicit and symbolic state-space generation techniques and symbolic CTL model-checking algorithms are available. For stochastic and timing behavior, sparse-storage and Kronecker-based numerical solution approaches are available when the underlying process is a Markov chain, and discrete-event simulation is available for any type of underlying process. In addition, certain classes of non-Markov models can be solved numerically. For more details, see G. Ciardo et al., “Logical and stochastic modeling with SMART”, in Proc. Mod. Tech. and Tools for Comp. Perf. Eval., LNCS 2794, Springer, 2003, or the SMART User Manual available at http://www.cs.ucr.edu/~ciardo/SMART/.

Overview  Input to SMART can contain many interacting parametric models, each defining a variety of measures and expressed in the most appropriate formalism. Petri nets and discrete or continuous time Markov chains (DTMCs and CTMCs) are currently available, and a software-oriented formalism is planned. Models interact: a measure computed in one model can be an input parameter for another model. For logical analysis, SMART generates the (reachable) state space of a model using highly optimized algorithms ranging from explicit ones based on “flat” hashing or search trees or “structured” trees of trees, to symbolic ones based on multi-way decision diagrams (MDDs). For the symbolic encoding of the transition relation specifying the states reachable from each state in one step, SMART uses either a boolean sum of Kronecker products of (small) boolean matrices, or a matrix diagram. For mostly asynchronous systems, this encoding is enormously more efficient in terms of memory and it allows us to exploit the inherent event locality to greatly improve the runtimes by using the sophisticated saturation algorithm. SMART uses these techniques to improve the efficiency of CTL model checking as well.

For stochastic timing analysis, a numerical solution is feasible if the underlying process is a DTMC, CTMC, or a semi-regenerative process satisfying certain conditions. If the distributions in a model are all geometric or all exponential, the underlying process is clearly a DTMC or a CTMC, and SMART can employ an explicit approach requiring memory proportional to the number of arcs in the Markov chain. For CTMCs, SMART can also employ a memory-efficient implicit Kronecker or matrix-diagram encoding. In the semi-regenerative case, an embedded DTMC is built by computing the expected state holding times and state-to-state transition probabilities from a set of subordinate processes; currently, SMART requires these to be CTMCs, but we plan to extend the approach to subordinate processes solved using discrete-event simulation. In general, however, it is a priori undecidable whether the underlying process is semi-regenerative. SMART makes an efficient on-line classification of a process during its generation, by attempting to generate its embedded and subordinate processes. If it succeeds, it can employ the numerical solution; otherwise, SMART must rely on discrete-event simulation. Recognizing the embedded, or regeneration, instants is a fundamental capability for this approach, which SMART also exploits when using regenerative simulation. Alternatively, traditional batch-means simulation is available.

SMART Language  SMART uses a strongly-typed language with five types of basic statements (declaration, definition, model, expression, and option statements) and two compound statements (for and converge statements) that can be arbitrarily nested. The predefined types bool, int, bigint, real, and string are available in SMART. In addition, composite types can be defined using concepts of aggregate, set, and array, and a type can be further modified by a nature describing stochastic characteristics: const (non-stochastic), ph (random variable with discrete or continuous phase-type distribution), x and (arbitrary ran-

*This work was partially supported by the National Aeronautics and Space Administration under NASA Contract NAG-1-02095 and by the National Science Foundation under grants CCR-0219745 and ACI-0203971.
Random variables SMART can manipulate discrete and continuous phase-type distributions of which the exponential (exponential, geometric (geom), and non-negative integer constants are special cases. Combining ph types produces another ph type if phase-type distributions are closed under that operation. Internally, SMART uses an absorbing DTMC or CTMC to represent a ph int or ph real, respectively, but it builds it only if needed for a numerical computation. The number of states in the representation of a phase-type obtained through operators such as max and min can grow very rapidly. Mixing ph int and ph real or performing operations not guaranteed to result in a phase-type distribution forces SMART to consider the resulting type as generally distributed. Such general rand random variables can be manipulated only via simulation.

Modeling formalisms Components of a model are declared using formalism-specific types (e.g., the places of a Petri net). The model structure is specified via formalism-specific functions (e.g., in a Petri net, initialize the number of tokens in a place). Measures are user-defined functions that specify some constant quantity of interest (e.g., the expected number of tokens in a place in steady-state), and are the only components accessible outside of the model. Currently, the available formalisms are dtmc, ctmc, and spn (whose type of underlying stochastic process is determined by the distributions specified for the transitions).

State-space generation and storage The generation and storage of the state space of a model is a key component of any state-space-based solution technique and an integral part of model checking. SMART implements both explicit techniques, that store states individually in AVL trees, splay trees, or hash tables, and implicit (or symbolic) techniques that use multi-way decision diagrams (MDDs), an extension of binary decision diagrams (BDDs), to store sets of states. SMART uses a Kronecker or matrix diagram encoding of the transition relation between states, instead of a BDD encoding, to facilitate the detection and exploitation of the event locality inherently present in asynchronous systems.

The most efficient symbolic iteration strategy in SMART is saturation, which exhaustively fires, in an MDD node at level $k$, all enabled events that affect only levels $k$ and below, until the set encoded by the node reaches a fixed point. Subsequently, all the nodes below are also saturated, and thus need not be explored again, resulting in enormous reductions in computation time. Since only saturated nodes are stored, there is also a reduction in the peak storage requirements: often, the peak and final numbers of nodes are almost the same. This improves upon traditional approaches, in which the number of nodes can explode during generation, before “contracting” to the final representation.

CTL model checking CTL model-checking queries are available in SMART via a set of model-dependent measures with type stateset. Each stateset, a set of states satisfying a given CTL formula, is stored as an MDD, and all MDDs for a model instance are stored in one MDD forest, sharing common nodes for efficiency. All model-checking algorithms in SMART use symbolic techniques. The available functions can be grouped into atom builders (nostates, initialstate, reachable, and potential($\epsilon$), the potential states satisfying condition $\epsilon$); traditional set operators, CTL temporal logic operators, execution trace operators, and utility functions.

Kronecker encoding of the Markov chain matrix For spn models with expo or immediate transitions, SMART provides advanced solution methods based on a Kronecker or matrix diagram (an efficient structure that combines the idea of decision diagrams with Kronecker algebra) encoding of the underlying CTMC. These are quite effective in reducing the memory requirements and, like MDD methods, require a user-specified partition. Numerical algorithms (Jacobi, Gauss-Seidel, SOR) can be based on either the potential state space $\bar{S}$ or the actual state space $S$.

Approximations The converge statement specifies approximate model solutions based on a heuristic decomposition where submodels are solved in a fixed-point iteration. SMART executes the statements in the converge block until the variables declared in it change by less than some threshold $\epsilon$, relatively or absolutely, between iterations.

In addition, SMART provides a powerful approximation technique for stationary analysis of models having an underlying structured CTMC. By using an MDD encoding of $S$ and a Kronecker encoding of the CTMC, the technique performs an approximate aggregation for each level of the MDD. The transition rates between aggregated states are determined efficiently from the Kronecker structure. However, the probability of a state within its group may be needed and, since the rates of each aggregated CTMC may depend on the probabilities computed for the other aggregations, fixed-point iterations are used to break cyclic dependencies. The results can be quite accurate.