Linking *STeP* with SPIN

Anca Browne, Henny Sipma, Ting Zhang *

Computer Science Department
Stanford University
Stanford, CA 94305-9045
sipma@cs.stanford.edu

Abstract. We have connected *STeP*, the Stanford Temporal Prover, with SPIN, an LTL model checker. In this paper we describe the translation of fair transition systems into Promela, in particular how weak and strong fairness constraints are handled. The paper presents some preliminary experimental results using this connection.

1 Introduction

The Stanford Temporal Prover, *STeP*, supports the computer-aided formal verification of concurrent and reactive systems based on temporal specifications [BBC+95]. *STeP* combines *algorithmic* with *deductive methods* to allow for the verification of a broad class of systems, including parameterized (N-process) programs, and programs with infinite data domains. Systems are analyzed modularly [FMS98]: components and subsystems can be analyzed individually and properties proven over these components are then automatically inherited by systems that include them. This allows a selective use of tools appropriate for the module at hand.

In the original version of *STeP*, *STeP*1, we provide a full range of verification tools. Deductive tools include *verification rules*, which reduce simple temporal properties to first-order verification conditions [MP95], and an interactive theorem prover. Algorithmic tools include an explicit-state and a symbolic model checker, an integrated suite of decision procedures [Bjø98] that automatically check the validity of a large class of first-order formulas, and tools for invariant generation to support the deductive tools. *Verification diagrams* [MBSU98], which reduce the proof of arbitrary temporal properties to first-order verification conditions and an algorithmic model check, combine the deductive and algorithmic tools.

In the new version of *STeP*, *STeP*2, we are moving towards a more *open architecture*. Realizing that it is impossible and also undesirable to single-handedly support and further develop the full range of tools, we have decided to focus our efforts on methods for high-level proof construction,

* This research was supported in part by the National Science Foundation under grant CCR-98-04100 and CCR-99-00984 ARO under grants DAAH04-96-1-0122 and DAAG55-98-1-0471, ARO under MURI grant DAAH04-96-1-0341, by Army contract DABT63-96-C-0096 (DARPA), by Air Force contract F33615-99-C-3014, and by ARPA/Air Force contracts F33615-00-C-1693 and F33615-99-C-3014

including abstraction, modularity, diagrams, and hybrid system reduction, and take advantage of specialized, and highly optimized tools such as SPIN [Hol91,Hol97] to provide the algorithmic support.

In this abstract we describe the current interface between *STeP* and SPIN. As we have only recently started the integration, this work is still very much in progress; we are convinced that as we get more familiar with SPIN, many optimizations can be made. We also hope to benefit from input from more experienced SPIN users and developers.

2 Computational Model and Specification Language

In *STeP* we represent reactive systems as *fair transition systems* (FTS) [MP95]. A fair transition system $\langle \mathcal{V}, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ is given by a finite set of system variables $\mathcal{V}$, defining a state space Σ, an *initial condition* Θ, which is a subset of Σ, a set of *transitions* $\mathcal{T}$, each of which is a binary relation over Σ, describing how the system can move from one state to the next, and the *justice and compassion* requirements $\mathcal{J} \subseteq \mathcal{T}$, and $\mathcal{C} \subseteq \mathcal{T}$, respectively.

In our framework, we assume an *assertion language* based on first-order logic. Θ is expressed as an assertion over the system variables, and each transition τ is described by its *transition relation* $\rho_\tau(\mathcal{V}, \mathcal{V}')$, an assertion over $\mathcal{V}$ and a set of *primed variables* $\mathcal{V}'$ indicating their next-state values. We assume that $\mathcal{T}$ includes an *idling transition*, whose transition relation is $\mathcal{V} = \mathcal{V}'$.

A *run* of $\mathcal{S}$ is an infinite sequence of states $s_0, s_1, \ldots$, where s_0 satisfies Θ and for every s_i there is a transition $\tau \in \mathcal{T}$ such that (s_i, s_{i+1}) satisfy ρ_τ.

The fairness requirements state that just (or *weakly fair*) transitions $\tau \in \mathcal{J}$ cannot be continuously enabled without ever being taken. Compassionate (or *strongly fair*) transitions cannot be enabled infinitely often without being taken. Every compassionate transition is also just. A *computation* is a run that satisfies these fairness requirements.

Properties of systems are expressed as formulas in *linear-time temporal logic* (LTL). *Assertions*, or state-formulas, are first-order formulas with no temporal operators, and can include quantifiers. *Temporal formulas* are constructed from assertions, boolean connectives, and the usual *future* ($\Box, \Diamond, \bigcirc, \mathcal{U}, \mathcal{W}$) and *past* ($\boxminus, \diamondminus, \ominus, \mathcal{B}, \mathcal{S}$) temporal operators [MP95]. A *model* of a temporal property φ is an infinite sequence of states $s_1, s_2, \ldots$ that satisfies φ. For a system $\mathcal{S}$, we say that φ is *$\mathcal{S}$-valid* if all the computations of $\mathcal{S}$ are models of φ.

3 Translating FTS into Promela

To enable model checking with SPIN, the FTS must be translated into Promela, the system description language of SPIN. Since the definition of a transition system in *STeP* is very general, the translation is applicable only to a subset of transition systems, namely those (1) that are syntactically finite-state (that is, all datatypes are finite), and (2) whose transition relations are all of the form

$$\rho_\tau = \bigvee_i \rho_{\tau^i}$$

with τ^i being called a mode of τ, and

$$\rho_{\tau^i} = \mathit{enabled}(\tau^i) \ \wedge \ \bigwedge_{v \in \mathcal{V}} \mathit{action}(v)$$

where $\mathit{enabled}(\tau^i)$ is an assertion over unprimed variables, characterizing the set of states on which τ^i is enabled, and $\mathit{action}(v) : v' = e$, with e an expression over unprimed variables.
A transition system $\Phi = \langle \mathcal{V}, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ is translated into Promela by creating an initialization process for Θ, and one process for each transition $\tau \in \mathcal{T}$, as shown in Figure 1. The translation strategy reflects the intuition that a transition represents a single atomic process, and the modes of the transition correspond to the different activities of the process.

$$
\begin{array}{l}
\mathit{proctype}\ P_\tau ()\ \{ \\
\qquad\qquad \mathit{do} \\
\qquad\qquad ::\ \mathit{atomic}\ \{\mathit{enabled}(\tau^1) \rightarrow \mathit{action}(\tau^1);\ \} \\
\qquad\qquad \vdots \\
\qquad\qquad ::\ \mathit{atomic}\ \{\mathit{enabled}(\tau^n) \rightarrow \mathit{action}(\tau^n);\ \} \\
\qquad\qquad \mathit{od} \\
\qquad\quad \}
\end{array}
$$

Fig. 1. Translation from τ to $\mathcal{P}_\tau$

The translation of the *STeP* LTL specification to SPIN format is straightforward. SPIN automatically generates a *stuttering-closed* automaton from any future LTL formula without the $\bigcirc$ (next-state) operator.

3.1 Handling Fairness Constraints

In *STeP* transitions may be unfair, weakly fair or strongly fair. SPIN supports weak fairness at the level of the processes, and thus by translating each transition into a separate proctype, each transition is, by default, modeled as weakly fair.
Unfair transitions simulate possibly non-terminating statements. They are modeled in Promela by adding an empty statement to the transition process, as shown in Figure 2. Note that the *idling transition* τ_I is *unfair* and is included in every Promela program translated from an FTS.

$$
\begin{array}{l}
\textit{proctype } P_\tau\,()\ \{ \\
\quad\quad\quad\quad do \\
\quad\quad\quad\quad ::\ atomic\ \{enabled(\tau^1);\ \} \\
\quad\quad\quad\quad ::\ atomic\ \{enabled(\tau^1) \rightarrow action(\tau^1);\ \} \\
\quad\quad\quad\quad \vdots \\
\quad\quad\quad\quad ::\ atomic\ \{enabled(\tau^n);\ \} \\
\quad\quad\quad\quad ::\ atomic\ \{enabled(\tau^n) \rightarrow action(\tau^n);\ \} \\
\quad\quad\quad\quad od \\
\quad\quad\quad \}
\end{array}
$$

Fig. 2. Translation from τ to P_τ

Strong fairness states that if a transition is enabled infinitely often it must be taken infinitely often. This property can be expressed in LTL as

$$\Box \Diamond\, enabled(\tau) \rightarrow \Box \Diamond\, taken(\tau)$$

A convenient way to represent the predicate $taken(\tau)$ is by introducing a new global variable $t : [1 \ldots N]$, with $N = |\mathcal{T}|$, to $\mathcal{V}$ and to augment every transition $\tau_i \in \mathcal{T}$ (assuming an arbitrary order on $\mathcal{T}$) with the assignment $t' = i$. Now the predicate $taken(\tau_i)$ can be expressed by

$$taken(\tau_i) \Longleftrightarrow t = i$$

We now can incorporate the strong fairness requirements in the specification as follows:

$$\Phi = \left(\bigwedge_{\tau \in \mathcal{T}} \big(\Box \Diamond\, enabled(\tau) \rightarrow \Box \Diamond\, taken(\tau) \big) \right) \rightarrow \varphi$$

Note that the validity of safety properties is independent of the fairness requirements of the system, so for proofs of safety formulas (currently identified by a conservative syntactic check) the strong fairness constraints are omitted from the specification for obvious efficiency reasons.

4 Implementation and Preliminary Results

Implementation The current interface between *STeP* and SPIN is file-based. Upon clicking the SPIN button on the *STeP* user interface, the transition system is translated into Promela and stored in a file. Then SPIN is invoked to generate a never claim for the specification in another file, and SPIN is run on these two files, to generate the C-files. The C-files are compiled and the resulting file *pan* is executed, currently with search depth 10,000. The output of all these steps is collected in a log file, which, upon completion of *pan* is examined by *STeP* to determine the result of the model checking. There are three types of outcomes: (1) SPIN found a

Peterson's Algorithm (12 proctypes)			
Metric	Mutual Exclusion	Accessibility	One Bounded Overtaking
Formula Template φ	$\Box\neg(p \wedge q)$	$\Box(p \rightarrow \Diamond q)$	$\Box(p \rightarrow q_1 \mathcal{W} q_2 \mathcal{W} q_3 \mathcal{W} q_4)$
Automaton $\neg\varphi$ Size (lines)	9	11	163
Automaton $\neg\varphi$ Generation Time	0m0.01s	0m0.00s	7m25.04s
Verification Time	0m0.19s	0m0.33s	0m0.22s
Verification Result	True	True	True

Semaphores (10 proctypes)			
Metric	Mutual Exclusion	Accessibility	One Bounded Overtaking
Formula Template φ	$\Box\neg(p \wedge q)$	$\Box(p \rightarrow \Diamond q)$	$\Box(p \rightarrow q_1 \mathcal{W} q_2 \mathcal{W} q_3 \mathcal{W} q_4)$
Automaton $\neg\varphi$ Size (lines)	9	166	163
Automaton $\neg\varphi$ Generation Time	0m0.01s	1m11.78s	6m55.91s
Verification Time	0m0.08s	0m0.50s	0m0.10s
Verification Result	True	True	False

Dining Philosophers(6 philosophers, 42 proctypes)			
Metric	Mutual Exclusion	Accessibility	One Bounded Overtaking
Formula Template φ	$\Box\neg(p \wedge q)$	$\Box(p \rightarrow \Diamond q)$	$\Box(p \rightarrow q_1 \mathcal{W} q_2 \mathcal{W} q_3 \mathcal{W} q_4)$
Automaton $\neg\varphi$ Size (lines)	9	-	163
Automaton $\neg\varphi$ Generation Time	0m0.00s	-	7m17.63s
Verification Time	3m58.94s	-	0m0.66s
Verification Result	True	-	False

*Above results obtained using SPIN 3.3.10 on Sun Ultra 2 with Solaris 2.6

Table 1. Experiment Results

counterexample, in which case it generates a file *step.trail*, (2) the search depth is exceeded, or (3) the property is valid. The current translation only applies to unparameterized transition systems. We are currently extending it parameterized transition systems with a fixed number of processes.

Preliminary Experimental Results We tested our current implementation on some typical properties (*mutual exclusion, accessibility and 1-bounded overtaking*) for three classic concurrent programs (*Semaphores, Peterson's algorithm and Dining Philosophers*) [MP95]. The results are shown in Table 1. Note that *accessibility* is a liveness property while the other two are safety properties. In the *Semaphores* case, the increase of automaton size for accessibility is due to the incorporation of two *strong fairness* conditions. For the *Dining Philosophers* [1] case, with 12 *strong fairness* conditions, the automaton could not be constructed, because it was too large.

References

[BBC+95] N.S. Bjørner, A. Browne, E.S. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP:

[1] There are six philosophers in total, each of which is represented by seven transitions. Of the seven transitions, two are semaphore requests which are compassionate. See [MP95] page 199 for details.

The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, November 1995. available from `http://www-step.stanford.edu/`.

[Bjø98] N.S. Bjørner. *Integrating Decision Procedures for Temporal Verification.* PhD thesis, Computer Science Department, Stanford University, November 1998.

[FMS98] B. Finkbeiner, Z. Manna, and H.B. Sipma. Deductive verification of modular systems. In W.P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference, COMPOS'97*, vol. 1536 of *Lecture Notes in Computer Science*, pages 239–275. Springer-Verlag, December 1998.

[Hol91] G. Holzmann. *The Design and Validation of Computer Protocols.* Prentice Hall Software Series. Prentice Hall, 1991.

[Hol97] G. Holzmann. The model checker spin. *IEEE Transaction on Software Engineering*, 23(5):279–295, May 1997.

[MBSU98] Z. Manna, A. Browne, H.B. Sipma, and T.E. Uribe. Visual abstractions for temporal verification. In A. Haeberer, editor, *Algebraic Methodology and Software Technology (AMAST'98)*, vol. 1548 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, December 1998.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.