next field of second version. This split corresponds to the fact that $p\mathcal{U}q \equiv q \vee (p \wedge \bigcirc p\mathcal{U}q)$. (The 'nexttime' operator '$\bigcirc$' is not supported in the translation, as it is not compatible to the partial order reduction [3] implemented in SPIN and its use in concurrent programs has been challenged [4]).

Whenever there are no new formulas, the node is added to the list of currently existing nodes if a node with the same subformulas does not exist there already. Then, a successor node is generated, with an edge from the old one. The successor has initially its *new* subformulas set to the *next* subformulas of the old node, and an empty set of *old* and *next* subformulas. If such a node does already exists in the list of existing nodes, the old version is updated by adding new incoming edges from the new version. Initially, one starts the algorithm with a node that has the formula to be translated as its only *new* subformula, one incoming edge from the dummy node *init*, and no *old* or *next* subformulas. The details of the algorithm and its correctness proof can be found in [1].

At the end, each node which has an edge from the dummy *init* node is an *initial node*. An execution of the automaton starts from an initial state and passes nodes which agree with the input on the values of atomic predicates $(p, q, \ldots)$. An execution *accepts* iff for each subformula of the form $p\mathcal{U}q$, it passes infinitely often either states that contain q or states that do not contain $p\mathcal{U}q$.

References

[1] R. Gerth, D. Peled, M. Vardi, P. Wolper, Simple On-the-fly Automatic Verification of Linear Temporal Logic, Protocol Specification Testing and Verification, 1995, Warsaw, Poland,

[2] G. J. Holzmann, Design and Validation of Computer Protocols, Prentice Hall Software Series, 1992.

[3] G. J. Holzmann, D. Peled, An Improvement in Formal Verification, 7*th* International Conference on Formal Description Techniques, Berne, Switzerland, 1994, Chapmann & Hall, 197–211.

[4] L. Lamport, What good is temporal logic, in R.E.A. Mason (Ed.), Information Processing 83, Elsevier Science Publishers, 1983, 657-668.

Checking Linear Temporal Logic Properties

Doron Peled and Gerard J. Holzmann
AT&T Bell Laboratories
600 Mountain Avenue, Murray Hill, NJ 07974, USA

A new version of SPIN is described, in which standard linear temporal logic (LTL) notation, as developed by Manna and Pnueli, can be used for expressing properties of PROMELA models. This allows asserting properties of concurrent programs using the LTL operators such as $\Diamond$ (eventually), $\Box$ (always) and $\mathcal{U}$ (until), combined using the usual Boolean operations (and, or, not, implies). For example, the property $\Box\Diamond p \rightarrow \Diamond q$ means that if p happens always eventually, or in other words, infinitely often, then q will eventually happen.

An efficient algorithm [1] translates LTL formulae directly into PROMELA's 'never claims', allowing immediate automatic verification by SPIN. Although in general the translation gives an exponential automaton, the algorithm described in [1] performs quite well on the temporal formulas typically encountered in verification.

The algorithm uses a data structure that includes six fields for each automaton node:

1. The node's *name*.
2. The list of *incoming* edges from other nodes.
3. *New* unprocesses subformulas.
4. *Old* processes formulas.
5. Subformulas for the next node.

The algorithm takes each *new* subformula in its turn and checks its main logical operator. Depending on the operator, the algorithm can split the node to two or add subformulas to the *new* or *next* fields. Then the processes formula is moved from *new* to the *old* field (of both copies, if the node was splited). For example, if the subformula is $p\mathcal{U}q$, then the node will be splited, with q added as a *new* subformula to the first versions, and p is added as a *new* subformula to the second version. Also the subformula $p\mathcal{U}q$ is added to the