A Sequent Calculus for a First-order Dynamic Logic with Trace Modalities for Promela$^+$

Florian Rabe[1], Steffen Schlager[2], and Peter H. Schmitt[2]

[1] International University Bremen
[2] Universität Karlsruhe

Abstract. We introduce the first-order dynamic logic *DLP* for Promela$^+$, a language subsuming the modelling language Promela of the Spin model checker. In *DLP* trace modalities can be used to reason about the temporal properties of programs. The definition of *DLP* includes a formal semantics of the Promela$^+$ language. A sound and relatively complete sequent calculus is given, which allows deductive theorem proving for Promela$^+$. In contrast to the Spin model checker for Promela, this calculus allows to verify infinite state models. To demonstrate the usefulness of our approach we present two examples that cannot be handled with Spin but that can be derived in our calculus.

1 *DLP*—Dynamic Logic for Promela$^+$

Dynamic Logic (DL) [4, 5] is an extension of first-order predicate logic with modalities $[\pi]F$ and $\langle\pi\rangle F$ for each program π of some programming language and DL formula F. DL allows to reason about the input/output behaviour of a program. However, sometimes it is desirable to reason about intermediate states of a program as well. This becomes possible if DL is extended with additional, so-called trace modalities $[[\pi]]F$, $[\langle\pi\rangle]F$, $\langle\langle\pi\rangle\rangle F$, and $\langle[\pi]\rangle F$, as shown in [1].

The programming language we consider in this paper is Promela$^+$ whose syntax is essentially the same as of Promela [7], the modelling language of the model checker Spin [6]. Besides the usual constructs like assignments, loops, etc. Promela offers dynamic process creation, synchronous and asynchronous communication through channels, and non-deterministic choice. Due to lack of space we cannot present the formal syntax here.

The semantic domain of DL are extended Kripke structures. The states are first-order structures (all sharing the same universe) and for each elementary command there is one transition relation reflecting the semantics of the considered programming language. A trace of a program π for an initial state s is the (possibly infinite) sequence of states of a possible run of π starting in s. Note, that for non-deterministic languages like Promela the semantics of a program is a set of traces for each initial state.

In order to reason about non-terminating programs we apply induction on the length of the traces. This requires the introduction of restricted programs π^t for a program π and a (positive) integer term t.[1] Assuming S is the set of traces of π and n is the denotation of t, the intended semantics of π^t is the set of all initial segments of traces in S whose length is at most n.

Formula $[\pi]F$ ($\langle\pi\rangle F$) holds in state s iff *for all* (there *exists* a) possible end state(s) of π when started in s the formula F holds. Formula $[[\pi]]F$ ($[\langle\pi\rangle]F$) holds iff *for all* (there *exists* a) state(s) on *all* traces of π the formula F holds. Formula $\langle[\pi]\rangle F$ ($\langle\langle\pi\rangle\rangle F$) holds iff there exists a trace of π on which *for all* (there *exists* a) state(s) F holds. Formulas not containing modalities are interpreted as usual.

For an informal semantics of Promela programs we refer to [7, 6]. A detailed formal semantics Promela$^+$ and therefore of Promela and can be found in [8]. In contrast to Promela, Promela$^+$ is not restricted to finite models. E.g., it is possible to create an unbounded number of processes, integer variables are not range restricted, and, most important, the initial state of a system may be (partially) unknown.

[1] These definitions are not present in [8], but we intend to publish them in an upcoming paper.

2 A Sequent Calculus for *DLP*

We can only present two very characteristic rules as examples (as usual, the semantics of a rule is that the validity of the premisses above the line implies the validity of the conclusion):

$$\frac{ex(c) \vdash F \qquad ex(c) \vdash \mathit{eff}(c)[[\mathit{remProg}(\pi, i)]]F}{\vdash [[i : \pi]]F}$$

The purpose of this rule is to execute the first command c of the i-th process of π. Here i is a tag specifying that the i-th process of π has been non-deterministically scheduled for execution. This rule can only be applied if scheduling rules have been applied before that have introduced all possible tags, thus spawning branches in the proof for every possible scheduling decision. Also, if c is a composed command other rules have to be applied first decomposing it into elementary commands. This is necessary since only elementary commands have a well-defined effect on the global state (due to the non-deterministic interleaving of processes).

While tags are syntactic entities of *DLP*, *ex*, *eff*, and *remProg* are meta level abbreviations that allow to state several rules in one compact rule scheme: $ex(c)$ is a first-order formula that expresses the executability of c, $\mathit{eff}(c)$ expresses the state transitions caused by the execution of c and modifies the state in which the following formula is to be evaluated, and $\mathit{remProg}(\pi, i)$ is the program that remains to be executed after c has been executed. These abbreviations must be defined for all elementary commands c, e.g., if c starts a new process, $\mathit{remProg}(\pi, i)$ removes c from π and adds the new process instance to π.

Having the intuitive meaning of these functions in mind it is easy to understand the rule: If c is executable, the formula $[[i : \pi]]F$, which states that F holds in all states on all traces, is reduced to F, which states that F holds in the current state, and $\mathit{eff}(c)[[\mathit{remProg}(\pi, i)]]F$, which states that F holds in all states on all traces that arise if the remaining program is executed in the next state (characterised by the state update $\mathit{eff}(c)$). If c is not executable, the proof goal is closed immediately.

Secondly, the following is an example for a rule that introduces the programs π^t in order to use induction on t:

$$\frac{\vdash \forall t : int.t \geq 1 \rightarrow [[\pi^t]]F}{\vdash [[\pi]]F}$$

2.1 Soundness and Relative Completeness

Soundness has to be shown separately for each rule. The proofs are technical, but not difficult. They can be found in [8].

The basic idea behind the (relative) completeness proof is to show by Gödelisation that *DLP* is not more expressive than first-order logic with arithmetic (see [8] for details). Relative completeness means that all valid formulas could be derived in the calculus if an oracle for arithmetic was available, i.e., a rule scheme providing all valid arithmetic formulas as axioms. Of course, in reality such a rule cannot exist but this is not harmful to "practical completeness". Rule sets for arithmetic are available, which—as experience shows—allow to derive all valid first-order formulas that occur during the verification of actual programs. Moreover, many arithmetic formulas can be automatically discharged by external decision procedures like CVC [9] and the Simplify tool, which is part of ESC/Java [3].

2.2 Examples

We now present two examples. Although they are extremely simple they cannot be verified using the model checker Spin, whereas their deductive verification can be done in a standard way. This shows the fundamental advantages of the deductive approach.

For these examples, note that in Promela **do**...**od** denotes a guarded non-deterministic choice, that is repeated until a break is encountered. Consider the program π defined as

```
do
  :: skip;
  :: x=0; break;
od
```

We now want to verify that $x \doteq 0$ holds in all possible final states of π. In *DLP* this property is expressed as $[\pi]x \doteq 0$. Spin cannot handle this because infinitely many and arbitrarily long runs exist for this model. However, using induction on t in π^t this model can be easily verified deductively.

For the second example let π be defined as

```
do
  :: x != 0; x=x−1
  :: else; break
od
```

where x is decreased as long as it is non-zero, and if x is zero, the loop terminates. We want to prove validity of the formula $\forall x : int.x \geq 0 \rightarrow [\pi]x \doteq 0$ expressing that for every initial value of x greater than 0 and all terminating runs $x \doteq 0$ holds in the final state. This property cannot be verified with Spin since it does not allow arbitrary initial states. However, Spin can easily verify a similar property with a fixed value for x.

While arbitrary initial states are not provided for in Promela they naturally occur in many realistic scenarios, e.g., when a program is started in the final state of another program. The initial state of a *DLP* verification is always arbitrary. If only certain initial states are to be considered restrictions must be included in the property to be verified.

The formal proof of this example is done by induction on x and can be found in [8]. Note that the induction hypothesis has to be specified interactively which can be very hard to find in practice.

3 Related Work and Conclusions

There are some approaches to define the semantics of Promela formally, most notably [2]. But the semantics that is induced by our semantics for Promela$^+$ is by far the most comprehensive one. Relative to this semantics we introduced a calculus that allows deductive theorem proving for Promela. No previous work in this direction exists for Promela or other non-deterministic multi-process languages.

The given examples show that *DLP* increases the set of verifiable Promela models significantly. As a minor drawback *DLP* has only six temporal operators to make statements about traces whereas Spin allows to specify arbitrary LTL formulas. E.g., in order to express the property "A holds for a while, and then B holds forever" *DLP* must be extended by a specific modality whereas this can be easily expressed in LTL.

References

1. B. Beckert and S. Schlager. A sequent calculus for first-order dynamic logic with trace modalities. In *International Joint Conference on Automated Reasoning*, volume 2083 of *LNCS*, pages 626–641, 2001.
2. M. del Mar Gallardo, P. Merino, and E. Pimentel. A generalized semantics of Promela for abstract model checking. *Formal Aspects of Computing*, 16(3):166–193, 2004.
3. ESC/Java (Extended Static Checking for Java). `http://research.compaq.com/SRC/esc/`.
4. D. Harel. *First-order Dynamic Logic*, volume 68 of *LNCS*. Springer, 1979.
5. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
6. G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
7. Promela Language Reference. Available at `http://spinroot.com/spin/Man/promela.html`.
8. F. Rabe. A dynamic logic with temporal operators for Promela. Master's thesis, Universität Karlsruhe, 2004. Available online at `http://i12www.ira.uka.de/~frabe/DLTP.pdf`.
9. A. Stump, C. W. Barrett, and D. L. Dill. CVC: A Cooperating Validity Checker. In E. Brinksma and K. G. Larsen, editors, *14th International Conference on Computer Aided Verification (CA V)*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer-Verlag, 2002.