Formal Verification of SystemCFL Specifications Using SPIN

Ka L. Man
Department of Mathematics and Computer Science
Eindhoven University of Technology
The Netherlands

Abstract: The formal language SystemCFL is the formalization of SystemC. The language semantics of SystemCFL was formally defined in a standard structured operational semantics (SOS) style. For verification purposes, in this paper, we present an approach to use the SPIN model checker as a verification engine for SystemCFL specifications, by translating SystemCFL specifications to PROMELA that is the input language of SPIN. We illustrate the practical interest of our approach with a case study: a hazardous circuit.

Key–Words: SystemC, SystemCFL, formal semantics, formal verification, PROMELA, SPIN model checker

1 Introduction

SystemC [1] is a modeling language (without formal semantics) consisting of C++ class libraries and a simulation kernel for designs at the system-behavioral and RTL. Recently, SystemC has received an extreme increase in industrial acceptance for system specification and simulation.

The goal of developing a formal semantics is to provide a complete and unambiguous specification of the language. It also contributes significantly to the sharing, portability and integration of various applications in simulation, synthesis and formal verification. To develop a formal semantics for a portable subset of SystemC, the formal language SystemCFL of SystemC was developed by [2], [5] and [7]. The main goal of SystemCFL is to provide the formal reasoning of SystemC designs and the formal analysis of the behavior of SystemC processes.

Since processes are the basic units of execution within SystemC that are used to simulate the behavior of a device or a system, Process Algebra [3] was chosen as the mathematical framework for SystemCFL. Process algebra was used, because it provides an elegant notation for transition, and allows for axiomatic reasoning.

Based on the informal semantics presented in [1], the language semantics of SystemCFL was formally defined in a standard structured operational semantics (SOS) style [4].

Recently, through some case studies: synchronous D flip flop, memory and remote procedure call (RPC) protocol, it was illustrated in [5] that SystemCFL can be reasonably efficiently used to model software, hardware and concurrency.

For verification purposes, in this paper, we present an approach to use the SPIN model checker ([9] and [10]) as a verification engine for SystemCFL specifications, by translating SystemCFL specifications to PROMELA [11] that is the input language of SPIN.

Among various formal verification tools, the SPIN model checker was chosen, because it is one of the most successful software tools that can be used for the formal verification of distributed software systems and hardware. Furthermore, the input language of the SPIN model checker is PROMELA that is a popular language for building verification models. It is widely used in industrial and academic fields. Also, PROMELA is similar to the language C. This makes PROMELA easy to understand by verification engineers, researchers and even students.

As related works, a formal translation was defined (in [6]) from SystemCFL to a variant (with very general settings) of timed automata [15]. The practical benefit of the formal translation from a SystemCFL specification to a timed automaton [15] is to enable verification of properties of SystemCFL specification using existing verification tools for timed automata, such as Uppaal [14]. However, specifications of timed automata are not always trivial and intuitive for users not having a computer science background. In addition, variants of timed automata are used for different verification tools for timed automata. Users are required to adapt manually the settings of the variant of timed automata proposed in [6] for various verification tools. Also, [8] reported that some desired properties of finite state systems described in SystemCFL can be fed into the SMV Model Checker [13] to verify them.
This paper is organized as follows. In Section 2, we review the syntax and formal semantics of System\textsuperscript{CFL}. A brief introduction of the SPIN model checker and PROMELA is given in Section 3. Section 4 is devoted to the straightforward translation from System\textsuperscript{CFL} to PROMELA. We illustrate the main ideas of the straightforward translation form System\textsuperscript{CFL} to PROMELA using the case study of the hazardous circuit in Section 5. The SPIN model checker is used in Section 5 to check the presence of hazard of the case study. Finally, some concluding remarks are made in Section 6.

2 Formal Language System\textsuperscript{CFL}

In this section, we give an overview of the formal language System\textsuperscript{CFL} proposed in [2] and [7]. For an extensive treatment of System\textsuperscript{CFL}, the reader is referred to those papers.

In order to define the semantics of System\textsuperscript{CFL} processes, we need to make some assumptions about the data types. Let $\text{Var}$ denote the set of all variables $(x_0, \ldots, x_n)$, $\text{Sct}$ denote the set of all possible two values (i.e. 0 or 1) bit of arbitrary string length, and $\text{Value}$ denote the set of all possible values $(v_0, \ldots, v_n)$ that contains at least $\mathbb{B}$ (booleans), $\mathbb{R}$ (reals) and elements from $\text{Sct}$. A valuation is a partial function from variables to values (e.g. $x_0 \mapsto v_0$). The set of all valuations is denoted by $\Sigma$. The set $\text{Ch}$ of all channels and the set $\text{S}$ of all sensitivity lists with clocks may be used in System\textsuperscript{CFL} processes as are assumed.

A process term $P$ in System\textsuperscript{CFL} is built from atomic process terms $AP$. System\textsuperscript{CFL} consists of various operators that operate on process terms. The formal language System\textsuperscript{CFL} is defined according to the following grammar:

$$AP ::= \delta \mid \text{skip} \mid x := e \mid \Delta e_n \mid \gg$

$$P ::= AP \mid P \bullet b \mid P \mid P \circ P \mid P \cdot P \mid P \Theta P \mid P \Delta_d P \mid P \diamond P \mid P * P \mid P \mid P \mid P || \ell P \mid P \sim P \mid \partial H(P) \mid \tau_\ell(P) \mid \partial(P)$$

The operators are listed in descending order of their binding strength as follows: \{$\circ$, $\bullet$, $\Delta$, $\Theta$, $\ast$, $\diamond$, $\Rightarrow$, $\bowtie$, $\Theta$, $||, \sim$\}. The operators inside the braces have equal binding strength. In addition, operators of equal binding strength associate to the left, and parentheses may be used to group expressions.

Below is a brief introduction of the formal language System\textsuperscript{CFL}. Due to reason of space, rules for operational semantics of System\textsuperscript{CFL} are not given in this paper. For those interested in more details, please read [2] and [7]. System\textsuperscript{CFL} has the following atomic process terms and operators:

- the deadlock $\delta$ is introduced as a constant, which represents no behavior;
- the skip process term performs the internal action $\tau$;
- the assignment process term $x := e$, which assigns the value of expression $e$ to $x$ (modeling a SystemC “assignment” statement);
- the delay process term $\Delta e_n$ is able to delay the value of numerical expression $e_n$;
- the unbounded delay process term $\gg$ (modeling a SystemC “wait” statement) may delay for a long time that is unbounded or perform the internal action $\tau$;
- the conditional composition $p \bullet b \bullet q$ operates as a SystemC “then_if_else” statement, where $b$ denotes a boolean expression and $p, q \in P$;
- the watching process term $b \circ P$ is used to model a SystemC “watching” statement;
- the timeout process term $p \Delta q$ (modeling a SystemC “time out” construct) behaves as $p$ if $p$ performs a time transition before an unit of time $d \in \mathbb{R}_{>=0}$; otherwise, it behaves as $q$;
- the sequential composition $p \circ q$ models the process term that behaves as $p$, and upon termination of $p$, continues to behave as process term $q$;
- the alternative composition $p \Theta q$ models a nondeterministic choice between process terms $p$ and $q$;
- the watchdog process term $p \diamond q$ behaves as $p$ during a period of time less than $d$, at the unit of time $d$, $q$ takes over the execution from $p$ in $p \diamond q$; if $p$ performs an internal cancel $\chi$ action, then the delay is canceled, and the subsequent behavior is that of $p$ after $\chi$ is executed;
- the repetition process term $\ast p$ (modeling a SystemC “loop” construct) executes $p$ zero or more times;
- the parallel composition $p || q$, the left-parallel composition $p || \ell q$ and the communication composition $p \sim q$ are used to express parallelism;
- the encapsulation of actions is allowed using $\partial H(p)$, where $H$ represents the set of all actions to be blocked in $p$;
- the abstraction $\tau_\ell(p)$ behaves as the process term $p$, except that all actions names in $\ell$ are renamed to the internal action $\tau$;
• the grouping of actions and executing them in one atomic step can be done by using $\mathcal{D}(p)$.

A SystemCFL process is a quintuple $\langle P, \Sigma, \Sigma, S, \chi \rangle$. We use the convention $\langle p, \sigma', \sigma, s, m \rangle$ to write a SystemCFL process, where $p$ is a process term; $\sigma, \sigma'$ are valuations; $s$ is a sensitivity list with clocks; and $m$ is a channel. The set of actions $A_r$ is defined as follows: $A_r = \{aa(x, v), s(m), r(m), com(m), \chi, \tau\}$, where $aa(x, v)$ is the assignment action (i.e. the value of $v$ is assigned to $x$), $s(m)$ is the parameterized send action, $r(m)$ is the parameterized receive action, $com(m)$ is the parameterized communication action between $s(m)$ and $r(m)$, $\chi$ is the internal cancel action and $\tau$ is the internal action.

We give a formal semantics for SystemCFL processes in terms of a Labelled Transition System (LTS). Three kinds of transition relations are defined for SystemCFL processes. An action transition $\langle p, \sigma', \sigma, s, m \rangle \rightarrow (p', \sigma, \sigma'', s, m)$ is that the process $\langle p, \sigma', \sigma, s, m \rangle$ executes the action $a \in A_r$, starting with the current valuation $\sigma$ (at the moment of the transition taking place) and by this execution $p$ evolves into $p'$; notice that $\sigma'$ represents the previous accompanying valuation of the process, and $\sigma''$ represents the accompanying valuation of the process after the action $a$ is executed. Similarly, a termination $\langle p, \sigma', \sigma, s, m \rangle \Rightarrow (\chi, \sigma, \sigma'', s, m)$ is that the process executes the action $a$ followed by termination.

A time transition $\langle p, \sigma', \sigma, s, m \rangle \sim (p', \sigma, \sigma'', s, m)$ is that the process $\langle p, \sigma', \sigma, s, m \rangle$ may idle during an unit of time $d$ and then behaves like $\langle p', \sigma, \sigma'', s, m \rangle$.

3 The Spin Model Checker and PROMELA

This section briefly introduces the SPIN model checker and the modeling language PROMELA.

SPIN is a software package that allows the simulation of a specification written in the language PROMELA. It accepts correctness claims specified in the syntax of standard Linear Temporal Logic (LTL) [12]. SPIN has been successfully applied to the verification of several types of properties, such as model checking of LTL formulas, verification of state properties, unreachable code, etc.

PROMELA is a modeling language to describe finite-state systems. It resembles the programming language C with CSP [16] features. Here, we give a short overview of the most important PROMELA statements that are relevant to this paper. For a complete description of the syntax and semantics of PROMELA, we refer the reader to [10]. PROMELA statements:

• skip. The empty statement, which means “do nothing”.

• assignment. For instance, $y = 7$, which means the value 7 is assigned to the variable $y$.

• if. An if statement has the following form:

```
if :: guard1 -> statement1
:: ...........
:: guardn -> statementn
```

It is executable if at least one guard holds. If more than one guard is enabled, a choice is made non-deterministically.

• do. A do statement has the same form as an if statement (the keywords if and fi are replaced by do and od respectively). It is similar to an if statement, except that the statement is executed repeatedly, until the control is explicitly transferred to outside of the statement.

• timeout. It becomes executable if there is no other statement in the system which is executable. It has no effect when it is executed.

• atomic. It can be used to group statements into an atomic sequence; all statements are executed in one step. It is also used to initialize a series of processes in such a way that none of them can start executing statements until initialization of all of them has been completed.

• run. It is used to create new process.

4 Translation

This section presents a straightforward translation of a reasonable subset (that is relevant to this paper) of SystemCFL to PROMELA. Since SPIN and PROMELA are not specifically designed to cover timed behavior, the atomic process terms and operators of SystemCFL dealing with timing parameters are not translated. In addition, due to reason of space, the translation of the SystemCFL operators relating to the communication is also not shown in this section.

Intuitively, the SystemCFL deadlock $\delta$, the SystemCFL atomic process terms skip and assignment (of the form $x := e$) are translated into the deadlock, the statements skip and assignment (of the form $x = 7$) in PROMELA respectively.
The SystemCFL alternative composition \((p\Theta q)\) models a non-deterministic choice between process terms that can be translated to an if\(\_f\) statement in PROMELA. Similarly, the SystemCFL repetition \((sp)\) can be translated to a do\_od statement in PROMELA.

The SystemCFL grouping \((\odot(p))\) is defined in an exact way as the PROMELA statement atomic. Obviously, the SystemCFL grouping \((\odot(p))\) is translated to the PROMELA statement atomic.

Roughly speaking, the SystemCFL sequential composition \((p \bullet q)\) is translated to a PROMELA expression of the form \(\text{Statement1 ; Statement2}\), where the statements \(\text{Statement1}\) and \(\text{Statement2}\) describe the behavior of the process terms \(p\) and \(q\).

Also, the SystemCFL conditional composition of the form \(p \triangleleft b \triangleright \delta\) is translated to a PROMELA guard statement of the form \(\text{guard1}\rightarrow\text{Statement1}\), where \(b\) is equivalent to \(\text{guard1}\) and the statement \(\text{Statement1}\) describes the behavior of the process term \(p\).

The SystemCFL parallel composition \((p \parallel q)\) is translated to parallel run processes in PROMELA (e.g. \(\text{run process1()} ; \text{run process2()}\)).

Table 1 depicts the straightforward translation of a reasonable subset of SystemCFL to PROMELA presented in this section.

<table>
<thead>
<tr>
<th>SystemCFL</th>
<th>PROMELA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\delta)</td>
<td>deadlock</td>
</tr>
<tr>
<td>skip</td>
<td>skip</td>
</tr>
<tr>
<td>(x := e)</td>
<td>(x = 7)</td>
</tr>
<tr>
<td>(p\Theta q)</td>
<td>if(_f)</td>
</tr>
<tr>
<td>(*p)</td>
<td>do_od</td>
</tr>
<tr>
<td>(\odot(p))</td>
<td>atomic</td>
</tr>
<tr>
<td>(p \bullet q)</td>
<td>(\text{Statement1 ; Statement2})</td>
</tr>
<tr>
<td>(p \triangleleft b \triangleright \delta)</td>
<td>(\text{guard1}\rightarrow\text{Statement1})</td>
</tr>
<tr>
<td>(p \parallel q)</td>
<td>(\text{run process1()} ; \text{run process2()})</td>
</tr>
</tbody>
</table>

Table 1: Straightforward Translation.

5 Case study

Many case studies of the application of SPIN to the verification of software were already appeared in the literature [9]. In the section, we present the application of SPIN to verify a hazardous circuit (hardware design) described in SystemCFL, by translating such a hazardous circuit to PROMELA. The case study of the hazardous circuit is taken from [17], rather than devised by us.

5.1 Hazardous Circuit

Fig. 1 shows a circuit that contains a hazard. The output of the circuit has the logical function \(z = \tilde{a}\tilde{c}d + bcd\). Depending on the delays of the inverter and wires, during a transition on signal or wire \(e\), a spike may occur.

For instance, while \(\tilde{e}\) is changing from “1” to “0”, the other input signals are still constant. Hence, this leads a hazard at the output \(z\).

![Figure 1: A hazardous circuit.](image)

5.2 Hazardous Circuit in SystemCFL

We first write the process terms \(\text{AND}, \text{OR}\) and \(\text{INV}\) for describing the behavior of the AND gate, OR gate and inverter shown in Fig. 1 respectively.

In order to increase the readability, we introduce syntactic sugars for various process terms. The syntactic sugars for the process terms \(\text{AND}, \text{OR}\) and \(\text{INV}\) are as follows:

\[
\text{AND}(e, d, f, h) \equiv h \leftarrow e \land d \land f \land h \neq e \land d \land f \land \delta, \\
\text{OR}(e, d, h) \equiv h \leftarrow e \lor d \lor h \neq e \lor d \lor \delta, \\
\text{INV}(g, h) \equiv h \leftarrow 1 \land h \neq 1 \land g \land \delta, \text{where } e, d, f, g, h \in \text{Var}.
\]

It is then followed by the circuit implementation (i.e. connections of the gates). This can be done by defining the process term \(\text{circuit}\) that describes the behavior of the circuit shown in Fig. 1 The process term \(\text{circuit}\) has the following syntactic sugar:

\[
circuit \equiv \star ((\text{AND}(\tilde{a}, \tilde{c}, d, k_1) \Theta \text{INV}(\tilde{e}, c) \Theta \text{AND}(b, e, d, k_2) \\
\Theta \text{OR}(k_1, k_2, z)) \bullet \text{new } wsp := 0)
\]

The process term \(\text{circuit}\) indicates that one of the gates is selected non-deterministically at each transition. The use of the variable \(\text{new } wsp\) will be explained later.

Another process term \(\text{stimulus}\) is also defined. The aim of the process term \(\text{stimulus}\) is to give a new pattern input to the process term \(\text{circuit}\) at each transition by toggling one of the possible inputs. The syntactic sugar of the process term \(\text{stimulus}\) is as follows:
The variable $newsp$ is used as a flag to indicate that a new input pattern has been applied at the current transition. This technique implements the fundamental mode operation of the circuit, i.e., the change of the input is allowed only if the circuit is stabilized first.

The variable $old_z$ denotes the old value of $z$ before performing a transition. The variables $newsp$ and $old_z$ are needed for expressing correctness claims in SPIN. Since the process terms $circuit$ and $stimulus$ execute concurrently, the parallel composition is used to model the complete system. The complete system with initial value for variables is modeled as follows:

\[
\langle circuit \parallel stimulus, \sigma', \emptyset, \emptyset \rangle, \text{for some } \sigma', \text{ where } \\
\sigma = \{ a \mapsto 0, c \mapsto 0, b \mapsto 0, d \mapsto 0, newsp \mapsto 1 \}.
\]

### 5.3 PROMELA Model of the Hazardous Circuit

The PROMELA model of the hazardous circuit is as follows:

```promela
/*declaration*/
bit a, abar, b, c, cbar, d, z;
bit oldz;
bit k1, k2;
bit newsp;
#define AND(x,y,z,out)
(out != (x&&y&&z))->out = x&&y&&z
#define OR(x,y,out)
(out != (x||y))->out = x||y
#define INV(in,out)
(out != (1-in))->out = (1-in)
proctype circuit()
{
  do
    :: if
      :: AND(abar, cbar, d, k1);
      :: INV(cbar, c);
      :: AND(b, c, d, k2);
      :: OR(k1,k2,z);
    fi;
    newsp = 0
  od
}
proctype stimulus()
{
  do
    :: timeout ->
      atomic {
        newsp = 1;
        oldz = z;
        if
      }
  od
}
```

We do not further describe the construct of the PROMELA processes $circuit$ and $stimulus$, because they come from the straightforward translation (proposed in Section 4) from the SystemC to PROMELA. Notice that the syntactic sugars $AND$, $OR$ and $INV$ are translated into PROMELA macro definitions. The process $init$ is used to initialize variables and start processes. The statement $timeout$ is introduced and needed in the process $stimulus$ to model recovery actions from potential deadlock states.

### 5.4 Verification

The goal of the verification is to show the presence of hazard of the circuit shown in Fig. 1 using SPIN. We can express our claim using the special process $never$ claim in SPIN. The process $never$ claim is used to express behavior that should not happen. Our claim is as follows:

```promela
never{
  do
    :: skip
    :: ((newsp == 0) && (oldz != z))
    -> break /*transition*/
  od;
  do
    :: ((newsp == 1) && (oldz != z))
    :: ((newsp == 0) && (oldz == z))
    -> break /*spike*/
  od
}
```

The meaning of the proposition $((newsp == 0) && (oldz != z))$ is that no new input but there is a transition at the output. If this proposition holds, the first $do$-od loop breaks and moves to the second $do$-od loop. The keyword break exits a $do$ statement.
Note that the statement *skip* is essential. If it is omitted, the proposition must hold in the first reachable state. The is not what we want. We want to check the correctness regardless what the initial reachable state is. In the second *do-od* loop, at some nondeterministic time, one of the two conditions occurs: a new pattern is applied (i.e. newsp == 1), or the output changes back (i.e. oldz == 2) while the input remains the same. In the former, the circuit does not produce a hazard. In the latter, a hazard is found, which also means the never claim is matched and an incorrect behavior is detected. The above PROMELA model of the hazardous circuit was tested using SPIN, and a hazard was found.

6 Conclusion

In this paper, we first briefly reviewed the syntax and semantics of SystemCFL. Then, we introduced the SPIN model checker and its input language PROMELA. We also showed that a reasonable subset of SystemCFL can be easily translated into PROMELA. This result allows the application of SPIN to model check SystemCFL specifications. A case study (hazardous circuit) was given to show the effectiveness and applicability of our approach. As a future work, it would be interesting to investigate the use of discrete time PROMELA and SPIN [18] to verify SystemCFL specifications which depend on timing parameters.

Acknowledgements: The author would like to thank D.A. van Beek, P.J.L Cuijpers, M. Mousavi, M.A. Reniers, and R.R.H. Schiffelers for many stimulating and helpful discussions. The author is grateful to J.C.M. Baeten and J.E. Rooda for their support and encouragement.

References:


