tempo2hsal: Converting Tempo Models into HybridSal

Tool Description

Ashish Tiwari Bruno Dutertre

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

Report submitted under Honeywell subcontract No. C099110021 (P20795)

1 Introduction

This report describes the current implementation of a translator from Tempo (`.tioa`) to HybridSal (`.hsal`).

Tempo is a formal language for modeling distributed system as collections of timed input/output automata (TIOA). Timed I/O automata (TIOA) is a mathematical formalism for describing systems that involve both discrete and continuous dynamics. The *Tempo* toolset, developed by Veromod Inc.[1], supports system development based on TIOA specifications [ALL+08].

HybridSal is a language for modeling hybrid dynamical systems. Models in the HybridSal language can be analyzed using abstraction and model checking tools that are available as part of the HybridSal and SAL tool suites [Tiw12, Hyb03, dMOR+04].

Since both Tempo and HybridSal can describe compositions of hybrid state machines, it is natural to consider the possibility of translating models written in one formalism into the other formalism. This report describes a tool `tempo2hsal` that translates Tempo models into HybridSal models.

2 Tempo2HSal: Installation and Testing

The `tempo2hsal` tool is available as a tar gzipped file. The tool is installed by extracting the files from this archive and running

```
make install
```

The installation can be tested by running

```
make test
```

The installation requires `python`, `swig` and `dparser` be available on the system prior to the installation process. If these are unavailable, the `make install` script will provides links to resources from where these components can be downloaded and installed.

3 Tempo2HSal: Tempo to HybridSal Model Translator

The mapping between Tempo concepts and HybridSal concepts is shown in Table 1. The current version of the `tempo2hsal` translator works by mapping elements in the Tempo model (shown in Column 1) to elements of the HybridSal model (shown in Column 2).

[1] `http://www.veromodo.com/`

Tempo	HybridSal
Top Tempo Spec	Context
invariant p of m : e	`THEOREM p m |- G( e )`
vocabulary	declarations in outermost HybridSal context
automaton	Module
basicAutomaton	basemodule
composedAutomaton	asynchronous composition of modules
transitions	Guarded Commands
pre	guard
eff	RHS assignments of guaraded command
trajectories	guarded commands denoting differential equations
trajInvariant	guard
stop when	guard
evolve	RHS assignments of guaraded command
Enumeration [a,b]	`{ a, b }`

Table 1: Mapping from Tempo concepts to HybridSal concepts used to perform the translation

The translator can translate several Tempo models (present in the Tempo distribution) into HybridSal models. The `Makefile` lists some of the examples that the translator can successfull convert into HybridSal.

The `tempo2hsal` tool is implemented in python.

4 A Running Example

We show an example of a Tempo specification and the translated HybridSal model in this section. Due to space constraints, both the Tempo file and the generated HybridSal file are divided into two parts. The first part contains the model of the system. The second part contains the properties of the system.

Figure 1 presents the Tempo file describing an automaton `TTR`. This model is taken from the Tempo distribution. The invariants associated with TTR are shown in Figure 2.

The HybridSal file generated from the Tempo model and the invariants is shown in Figure 3 and Figure 4. It is easy to see the correspondence between Figure 1 and Figure 3, and again between Figure 2 and Figure 4.

Given the input file `twoTaskRace0.tioa`, the translater `tempo2hsal` creates the HybridSal file `twoTaskRace0.hsal`.

The generated HybridSal model can be automatically analyzed using HybridSal tools. In particular, the differential equations can be replaced by their conservative discrete abstraction using the HybridSal relational abstracter as follows:

```
automaton TTR(a1, a2, b1, b2: Real) where
  a1 > 0 /\ a2 > 0 /\ b1 >= 0 /\ b2 >= 0 /\ a2 >= a1 /\ b2 >= b1

  signature
    internal increment
    internal decrement
    output report
    internal set

  states
    count: Int := 0;
    flag: Bool := false;
    reported: Bool := false;
    now: Real := 0;
    first_main: DiscreteReal := a1;
    last_main: AugmentedReal := a2;
    first_set: DiscreteReal := b1;
    last_set: AugmentedReal := b2;

  transitions

    internal increment
      pre ~flag /\ now >= first_main;
      eff count := count + 1;
          first_main := now + a1;
          last_main := now + a2;

    internal set
      pre ~flag /\ now >= first_set;
      eff flag := true;
          first_set := 0;
          last_set := \infty;

    internal decrement
      pre flag /\ count > 0 /\ now >= first_main;
      eff count := count - 1;
          first_main := now + a1;
          last_main := now + a2;

    output report
      pre flag /\ count = 0 /\ ~reported /\ now >= first_main;
      eff reported := true;
          first_main := 0;
          last_main := \infty;

  trajectories
    trajdef traj
      stop when now = last_main \/ now = last_set;
      evolve d(now) = 1;
```

Figure 1: An example of Tempo specification taken from the Tempo distribution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Invariants

invariant of TTR:
  now >= 0;

invariant of TTR:
  now + b2 >= 0;

invariant of TTR:
  count >= 0;

invariant of TTR:
  reported => flag;

invariant of TTR:
  reported => count = 0;


invariant of TTR:
  first_main <= (a1 + now);

invariant of TTR:
  last_main >= now;

invariant of TTR:
  last_main ~= \infty =>
     (last_main <= (a2 + now));

invariant of TTR:
  ~reported <=> last_main ~= \infty;


invariant of TTR:
  first_main <= last_main;

invariant of TTR:
  first_set <= (b1 + now);

invariant of TTR:
  last_set >= now;

invariant of TTR:
  last_set ~= \infty =>
     (last_set <= (b2 + now));

invariant of TTR:
  ~flag <=> last_set ~= \infty;

invariant of TTR:
  ~flag =>  (Real)last_set + a2 >= ((Real)last_main);

invariant of TTR:
  ~flag => ((2 * a1) + (Real)last_set - (Real)first_main) >= 0;
```

Figure 2: An example of invariants for the Tempo specification in Figure 1 taken from the Tempo distribution.

```
twoTaskRace0: CONTEXT =
BEGIN

TTR: MODULE =
BEGIN
 LOCAL a1:REAL,a2:REAL,b1:REAL,b2:REAL
 OUTPUT count:INTEGER
 OUTPUT flag:BOOLEAN
 OUTPUT reported:BOOLEAN
 OUTPUT now:REAL
 OUTPUT first_main:REAL
 OUTPUT last_main:REAL
 OUTPUT first_set:REAL
 OUTPUT last_set:REAL
 INITIALIZATION
  count = 0 ;
  flag = FALSE ;
  reported = FALSE ;
  now = 0 ;
  first_main = a1 ;
  last_main = a2 ;
  first_set = b1 ;
  last_set = b2 ;
  a1 IN { a1 : REAL | a1 > 0 AND a2 > 0 AND b1 >= 0 AND b2 >= 0 AND a2 >= a1 AND b2 >= b1 } ;
  a2 IN { a2 : REAL | a1 > 0 AND a2 > 0 AND b1 >= 0 AND b2 >= 0 AND a2 >= a1 AND b2 >= b1 } ;
  b1 IN { b1 : REAL | a1 > 0 AND a2 > 0 AND b1 >= 0 AND b2 >= 0 AND a2 >= a1 AND b2 >= b1 } ;
  b2 IN { b2 : REAL | a1 > 0 AND a2 > 0 AND b1 >= 0 AND b2 >= 0 AND a2 >= a1 AND b2 >= b1 }

 TRANSITION
 [
 NOT(flag) AND now >= first_main  -->
    count' = count + 1 ;
    first_main' = now + a1 ;
    last_main' = now + a2
 []
 NOT(flag) AND now >= first_set  -->
    flag' = TRUE ;
    first_set' = 0 ;
    last_set' = 10000
 []
 flag AND count > 0 AND now >= first_main  -->
    count' = count - 1 ;
    first_main' = now + a1 ;
    last_main' = now + a2
 []
 flag AND count = 0 AND NOT(reported) AND now >= first_main  -->
    reported' = TRUE ;
    first_main' = 0 ;
    last_main' = 10000
 []
 TRUE AND NOT(now = last_main OR now = last_set)  -->
    nowdot' = 1
 ]
END ;
```

Figure 3: The HybridSal model generated by `tempo2hsal` when it is run on the Tempo example shown in Figure 1.

```
p1 : THEOREM
  TTR |- G(now >= 0);
p2 : THEOREM
  TTR |- G(now + b2 >= 0);
p3 : THEOREM
  TTR |- G(count >= 0);
p4 : THEOREM
  TTR |- G(reported => flag);
p5 : THEOREM
  TTR |- G(reported => count = 0);
p6 : THEOREM
  TTR |- G(first_main <= a1 + now);
p7 : THEOREM
  TTR |- G(last_main >= now);
p8 : THEOREM
  TTR |- G(last_main /= 10000 => last_main <= a2 + now);
p9 : THEOREM
  TTR |- G(NOT(reported) <=> last_main /= 10000);
p10 : THEOREM
  TTR |- G(first_main <= last_main);
p11 : THEOREM
  TTR |- G(first_set <= b1 + now);
p12 : THEOREM
  TTR |- G(last_set >= now);
p13 : THEOREM
  TTR |- G(last_set /= 10000 => last_set <= b2 + now);
p14 : THEOREM
  TTR |- G(NOT(flag) <=> last_set /= 10000);
p15 : THEOREM
  TTR |- G(NOT(flag) => last_set + a2 >= last_main);
p16 : THEOREM
  TTR |- G(NOT(flag) => 2 * a1 + last_set - first_main >= 0);
END
```

Figure 4: The properties in HybridSal model generated by `tempo2hsal` when it is run on the properties in Tempo shown in Figure 2.

```
bin/hsal2hasal examples/twoTaskRace0.hsal
```

Note that we are assuming here that the HybridSal relational abstracter tool [Tiw12] has been correctly installed.

The hybridsal relational abstracter generates a SAL file called `twoTaskRace0.sal`. We can use the SAL infinite bounded model checker and k-induction prover to verify the invariants in the SAL file. In particular, we note that several of the invariants are automatically proved by the k-induction prover. Some of the invariants need auxiliary lemmas for completing the proof. Some invariants, while true in the Tempo model, turn out to be false in the SAL file – this may be because of two reasons.

- The abstraction step (`hsal2hasal`) introduces spurious counter-examples
- The translater (`tempo2hsal` does not fully preserve the semantics of the Tempo model, but only generates a conservative HybridSal translation.

We discuss the latter issue in the next section.

5 Caveats

The Tempo to HybridSal translater attempts to preserve the semantics of the model during translation. However, in some cases, it does not capture the precise semantics of the Tempo model. In this section, we discuss some of the main Tempo constructs whose semantics may not be preserved during the translation.

Among the basic **types** supported by Tempo, some are not supported by HybridSal. The most prominent such type is `AugmentedReal`, which is the set of reals augmented with ∞. The current `tempo2hsal` translater maps `AugmentedReal` to `Real` in HybridSal, and maps ∞ to a fixed large real number. This translation is not semantic preserving. Similarly, `tempo2hsal` maps `discreteReal` to `Real`. HybridSal also does not natively support datatypes such as `queues`. It does support tuples and arrays, but the current version of the `tempo2hsal` translater does not translate any of these complex types.

Automaton **parameters** are also not handled in the most precise way currently. Tempo allows declaration of automaton with parameters, but also allows invariant properties to be stated on *uninstantiated* automata. The Sal and HybridSal language do not allow the use of a parametric module without creating an instance of it by giving proper actual arguments. In the current `tempo2hsal` tool, automaton parameters are, in some cases, mapped to local variables in the HybridSal module. The `where` constraint on the parameters in Tempo are used to provide *initial* values to these local variables in HybridSal. In this case, the HybridSal model includes behaviors of of the Tempo automaton *for all* possible instantiations of the parameters consistent with the `where` constraint.

Tempo has a `vocabulary` feature for defining namespaces. Automata can later import different vocabularies. There is no such feature in HybridSal. The current version of

the `tempo2hsal` translater simply includes all identifiers defined inside vocabularies inside the top HybridSal `context`. The translater then ignore all `import` statements later, since all defined identifiers are available anyway. Note that this can cause name conflicts, which can change the semantics of the model.

Other notable features missing in the translater include

- The `tempo2hsal` translater currently ignores all transition labels and events.
- Not all expressions in Tempo are translated into HybridSal correctly. For example, where expressions are not handled. Similarly, expressions that manipulate queues or tuples are also ignore presently.
- `initially` expressions are not translated
- The translation of composed automaton in Tempo to composed modules in HybridSal may not be semantic preserving, because the semantics of the composition operators are different.

We note here that the HybridSal model generated by `tempo2hsal` may not be the best possible HybridSal model for the given Tempo model. In particular, for improving the precision of analysis, HybridSal prefers to have all differential equations at "one place", rather than distributed across different modules.

Finally, we also note that when analyzing the HybridSal model using relational abstraction, the abstraction process also introduces spurious behaviors, some of which the user can eliminate by modifying the HybridSal model by hand.

Whenever the `tempo2hsal` tool is not sure if it has preserved the semantics of the Tempo model, it emits a **warning** message. The user should check if the generated HybridSal model is semantically equivalent to the input Tempo model.

The `tempo2hsal` tool is still incomplete in many ways, as noted above. However, it is a useful starting point for further development of a more comprehensive converter from Tempo to HybridSal. In particular, the tool is being made available in open source form for other users to modify and extend.

5.1 Implementation Details

The `tempo2hsal` tool is implemented in Python. The Tempo input file is parsed using dparser[2], and the HybridSal is generated through actions attached to the Tempo grammar. Specifically, the actions create an XML tree of the HybridSal output. The XML representation of HybridSal is then output in the usual HybridSal syntax by using a HybridSal pretty printer. The tool distribution contains sources for all these parts.

[2] `http://dparser.sourceforge.net/`

6 Conclusion

We have described the `tempo2hsal` tool that converts Tempo models into HybridSal models. We presented a concrete Tempo model and its translation generated by `tempo2hsal`. We also outlined the shortcomings and incompleteness in the tool.

References

[ALL+08] Myla Archer, Hongping Lim, Nancy A. Lynch, Sayan Mitra, and Shinya Umeno. Specifying and proving properties of timed i/o automata using tempo. *Design Autom. for Emb. Sys.*, 12(1-2):139–170, 2008.

[dMOR+04] L. de Moura, S. Owre, H. Ruess, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. SAL 2. In Rajeev Alur and Doron Peled, editors, *Computer-Aided Verification, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.

[Hyb03] Hybridsal: Modeling and abstracting hybrid systems, 2003. Computer Science Laboratory, SRI International, Menlo Park, CA. `http://www.csl.sri.com/users/tiwari/HybridSalDoc.ps`.

[Tiw12] A. Tiwari. Hybridsal relational abstracter. In *Proc. CAV*, volume 7358 of *LNCS*, 2012. `http://www.csl.sri.com/~tiwari/relational-abstraction/`.