Timed CSP Simulator

Marc Fontaine[1], Andy Gimblett[2], Faron Moller[2],
Hoang Nga Nguyen[2], and Markus Roggenbach[2]

[1] Heinrich-Heine-Universität Düsseldorf, Germany
[2] Swansea University, UK

1 Introduction

Time is an integral aspect of computer systems. It is essential for modelling a system's performance and also affects its safety or security. Timed Csp [5] conservatively extends the process algebra CSP with timed primitives, where real numbers ≥ 0 model how time passes with reference to a single, conceptually global, clock. While there have been approaches for model checking Timed Csp [1, 5], the simulation of Timed Csp was considered only recently [2, 6]. In this poster, we highlight the architecture and a number of selected features of our Timed Csp Simulator, which is a consolidated, mature version of the research prototype presented in [2].

2 Architecture

Timed Csp Simulator is an extension of the CSP animator within the open source tool ProB [3]. In Figure 1, we illustrate the architecture of the Timed Csp Simulator which consists of four main components: a Parser, a Timed Csp Interpreter, a Simulator and a GUI. In principle, the simulator works as follows: A Timed Csp specification is analysed by the Parser (written in Haskell) and translated to a representation in Prolog. This representation is passed into the Timed Csp Interpreter (written in Prolog). The Timed Csp Interpreter implements the "firing rules" of Timed Csp's operational semantics. Process states and the implementation of firing rules are then used by the Simulator for determining the set of actions available, the range of timed transitions as well as their corresponding resultant states. Finally, users interact with the Simulator through a GUI (written in Tcl/Tk) in order to control the simulation progress.

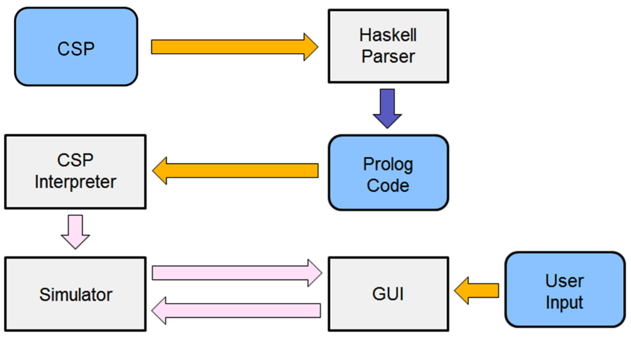


Fig. 1. Timed Csp Simulator's architecture

3 Features

Timed Csp Simulator is characterised by the following features:
Rational time: Timed Csp Simulator restricts processes to rational time only. This is reasonable as [4] proves that Timed Csp is closed under rational time, i.e., rational processes are closed under action transitions and rational delays. Prolog supports proper rationals, i.e., rational time can be expressed. In practice, the limitation to rational time turns out to be negligible. For instance, all examples of Schneider's book [5] can be dealt with in our simulator.
Separate firing rules: Although Timed Csp (as well as CSP) has a number of operators which can be treated as syntactic sugar, e.g., $\mathit{Wait}\ d = \mathit{Stop} \rhd^{d} \mathit{Skip}$, we follow the design of ProB where each supported (untimed or timed) operator has its own implementation of the corresponding firing rule. This results in a simulation without change of representation, where ProB can highlight in the specification text which process parts represent the current state.
Extensive set of operators: Compared to the language as given in [5], Timed Csp Simulator supports an extra set of untimed operators such as conditionals, untimed timeout and indexed external choice. To this end, we extend Timed Csp's operational semantics in [4].
Computing upper bounds of timed transitions: Based on the simulation theorem provided in [4], Timed Csp Simulator calculates the largest time step possible for a Timed Csp process in a recursive way. Consider, for instance, the process $T = (P \rhd^{e} Q) \rhd^{f} R$ with $0 < e < f$ and untimed processes P, Q and R. In T, the process P is enabled within the time interval $[0, e)$. A time step of length e (and a τ-transition) leads to the new state $Q \rhd^{f-e} R$, where P is not enabled anymore. Thus, the largest time step possible in T is e.
Automatic animation: Timed Csp Simulator supports two animation strategies, where the user selects the number of steps to be performed. **Random:** At each step of the animation, the simulator randomly selects an event or time step available from the interface. **Maximal progress:** At each step of the animation, the simulator selects an event or time step available from the interface in the following priority: (1) randomly select an external event, (2) select the internal event (3) select the maximal time step from a bounded interval, (4) randomly select a time step if arbitrary time steps are possible.
Backward compatibility: Timed Csp Simulator is backwards compatible for untimed CSP specifications. There are two ways to enable the Timed-CSP Simulator in ProB while opening specifications from files. **Explicit:** Files are named with the extension ".tcsp", or **Implicit:** Files (ended with the extension ".csp") contain any timed operator of delay event prefix, wait, timed timeout and timed interrupt.

4 Example

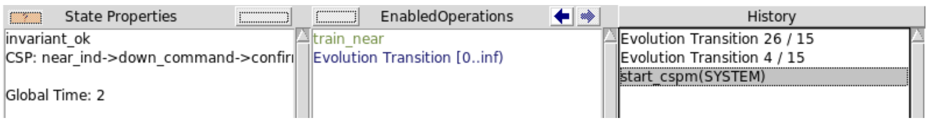
In order to illustrate the use of Timed Csp Simulator, we apply it to the well-known level crossing example [5]. This system consists of three components: a gate to block the traffic crossing the railway when a train is approaching, a controller to monitor the approach of trains and to instruct the gate to rise or lower appropriately, and a train passing by the crossing. Figure 2 presents the

$$\boxed{\begin{aligned}
GATE &= down.command \overset{100}{\rightarrow} down \rightarrow confirm \rightarrow GATE\\
&\quad \Box\ up.command \overset{100}{\rightarrow} up \rightarrow confirm \rightarrow GATE\\
TRAIN &= train.near \rightarrow near.ind \overset{300}{\rightarrow} enter.crossing \overset{20}{\rightarrow}\\
&\quad leave.crossing \rightarrow out.ind \rightarrow TRAIN\\
CONTROLLER &= near.ind \rightarrow down.command \rightarrow confirm \rightarrow CONTROLLER\\
&\quad \Box\ out.ind \rightarrow up.command \rightarrow confirm \rightarrow CONTROLLER\\
CROSSING &= CONTROLLER\,{}_{C}\|_{G}\,GATE\\
SYSTEM &= TRAIN\,{}_{T}\|_{C\cup G}\,CROSSING
\end{aligned}}$$

Fig. 2. Timed CSP's model of the level crossing.

Timed CSP's model of the level crossing as developed in [5]. It is straight forward to write this specification in the concrete syntax of Timed CSP Simulator.

In the following, we show two simulations of the level crossing example which highlight the *rational time only* and *automatic animation* features of Timed CSP Simulator. In Figure 3, we present a timed trace which includes two consecutive


State Properties
invariant_ok
CSP: near_ind->down_command->confirı

Global Time: 2

EnabledOperations
train_near
Evolution Transition [0..inf)

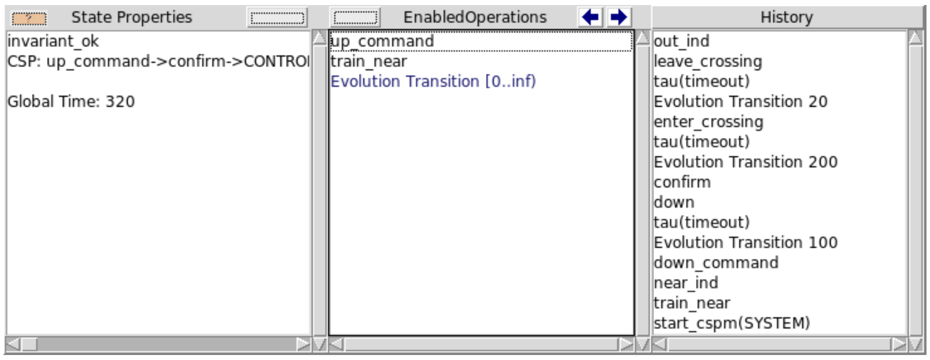
History
Evolution Transition 26 / 15
Evolution Transition 4 / 15
start_cspm(SYSTEM)

Fig. 3. A trace of two consecutive rational timed evolutions.

timed evolutions. The first timed evolution lasts for $\frac{4}{15}$ time unit while the second for $\frac{26}{15}$ time unit. After the two timed evolutions, the global time reaches $\frac{30}{15}$ time unit which is automatically converted into the simpler representation of 2 time units.

In the second simulation, rather than manually choosing an available action at each step of the simulation, we use the *automatic animation* feature to quickly generate a long timed trace of the level crossing. Figure 4 shows a timed trace generated by an animation of 15 transitions, following the *maximal progress* strategy. This timed trace illustrates the operation of the crossing as a train passing by. When the train approaches the crossing (by *train.near*), the controller requests the gate to move down (by *down.command*). The gate performs the action *down* and replies with a confirmation (by *confirm*) back to the controller. After the train has entered and exited the crossing, the controller is notified (by *out.ind*) so that it will instruct the gate to rise. At the end of this timed trace, the global time is 320 time units.

Timed CSP Simulator comes with an comprehensive test suite, derived from the fundamental algebraic laws of Timed CSP. A typical example is $P \rhd^5 (Q \rhd^3 R) = (P \rhd^5 Q) \rhd^8 R$, with $P = a \rightarrow Stop$, $Q = b \rightarrow a \rightarrow Stop$, and $R = c \rightarrow Stop$. Here, we check that simulations of the lhs are possible for the rhs and vice versa. Though these processes are not of much practical use, they highlight



invariant_ok
CSP: up_command->confirm->CONTROI

Global Time: 320

up_command
train_near
Evolution Transition [0..inf)

out_ind
leave_crossing
tau(timeout)
Evolution Transition 20
enter_crossing
tau(timeout)
Evolution Transition 200
confirm
down
tau(timeout)
Evolution Transition 100
down_command
near_ind
train_near
start_cspm(SYSTEM)

Fig. 4. A trace generated by automatic animation.

tricky features of the Timed Csp semantics and provide an argument that Timed Csp Simulator implements it correctly.

5 Conclusion

We have presented our tool Timed Csp Simulator, which is an extension of the CSP animator within ProB. We discussed architecture and features of the simulator. Besides simulating examples given in [5], we extensively use our tool within the SafeCap project[3] in order to explore how the change of signalling rules affects railway capacity. The ProB team has checked our implementation and made it available at `http://www.stups.uni-duesseldorf.de/ProB/index.php5/Download`. In the future, we plan to complete the simulator and to apply our tool within further application domains.

Acknowledgement We thank Erwin R. Catesbeiana (Jr.) for inspiring us to invest the extra time unit.

References

1. J. Dong, P. Hao, J. Sun, and X. Zhang. A reasoning method for Timed CSP based on constraint solving. *Formal Methods and Software Engineering*, 2006.
2. M. Dragon, A. Gimblett, and M. Roggenbach. A Simulator for Timed CSP. In *AVoCS'11*, 2011.
3. M. Leuschel. The ProB model checker. `http://www.stups.uni-duesseldorf.de/ProB/index.php5/Main_Page`.
4. F. Moller, H. N. Nguyen, and M. Roggenbach. Theoretical foundations for simulating Timed CSP. Technical report, Swansea University, In preparation.
5. S. Schneider. *Concurrent and real-time systems: the CSP approach.* Wiley, 2000.
6. T. Yamakawa, T. Ohashi, and C. Fukunaga. Development of an ML-based verification tool for Timed CSP processes. In *CPA'11*. IOS Press, 2011.

[3] SafeCap's website: `http://safecap.cs.ncl.ac.uk/`.