TMACS: a Tool for Modeling, Manipulation, and Analysis of Concurrent Systems

J. Jovanovski, M. Siljanoska, V. Carevski, D. Sahpaski, P. Gjorcevski, M. Micev, B. Ilijoski, and V. Georgiev

Institute of Informatics,
Faculty of Natural Sciences and Mathematics,
University "Ss. Cyril and Methodius",
Skopje, Macedonia
janejovanovski@gmail.com,maja.siljanoska@gmail.com,carevski@gmail.com,
dragan.sahpaski@gmail.com,pgorcevski@t-home.mk,metodi.micev@gmail.com,b.
ilijoski@gmail.com,vlatko.gmk@gmail.com

Abstract. This paper reports on an effort to build a tool for modeling, manipulation, and analysis of concurrent systems. The tool implements the CCS process language and can build labeled transition systems from CCS expressions. Furthermore, it can be used to reduce the state space of a labeled transition system and to check whether two labeled transition systems exhibit the same behaviour, using strong and weak bisimulation equivalence. The tool has the functionality needed to perform modeling, specification, and verification, illustrated on one classical example in the concurrency theory: the alternating bit protocol.

Keywords: CCS, labeled transition system, bisimulation equivalence, minimization, comparison, formal verification

1 Introduction

One of the process languages used to formally describe and analyse any collection of interacting processes is the Calculus of Communication Systems (CCS), process algebra introduced by Robin Milner and based on the message-passing paradigm [1][2]. CCS, like any process algebra, can be exploited to describe both specifications of the expected behaviours of the processes and their implementations [3]. The standard semantic model for CCS and various other process languages is the notion of labeled transition systems, a model first used by Keller to study concurrent systems [4]. In order to analyze process behaviour and establish formally whether two processes offer the same behavior, different notions of behavioral equivalences over (states of) labeled transition systems have been proposed [5].

One of the key behavioral equivalences is the strong bisimilarity [6] which relates processes whose behaviours are indistinguishable from each other [7]. Weak bisimilarity [1][8] is a looser equivalence that abstracts away from internal (non-observable) actions. These and other behavioral equivalences can be employed

to reduce the size of the state space of a labeled transition system and to check the equivalence between the behaviours of two labeled transition systems. This finds an extensive application in model checking and formal verification [9][5].

This paper presents TMACS, a tool that can be used to automate the process of modeling, specification, and verification of concurrent systems described by the CCS process language. TMACS stands for Tool for Modeling, Manipulation, and Analysis of Concurrent Systems. It is given as a Java executable jar library with very simple Graphical User Interface (GUI). TMACS can parse CCS expressions, generate labeled transition system from CCS in Aldebaran format [10], reduce a labeled transition system to its canonical form with respect to strong or weak bisimilarity and check whether two labeled transition systems are strongly or weakly bisimilar.

1.1 Related work

Different tools for modeling, specification and verification of concurrent reactive systems have been developed over the past two decades. Probably the most famous and most commonly used ones, especially in the academic environment, are the Edinburgh Concurrency Workbench (CWB) [11] and micro Common Representation Language 2 (mCRL2) [12][13].

CWB is a tool for analysis of cuncurrent systems, which allows for equivalence, preorder and model checking using a variety of different process semantics. Although CWB covers much of the functionality of TMACS and more, it has a command interpreter interface that is more difficult to work with, unlike our tool that has a Graphical User Interface (GUI), and is very intuitive. As far as we know CWB does not have the functionality for exporting labeled transition system graphs in Aldebaran format, that TMACS has.

mCRL2, the successor of μCRL, is a formal specification language that can be used to specify and analyse the behaviour of distrubuted systems and protocols. However, to our knowledge, mCRL2 does not provide the possibility to define systems' behaviour in the CCS process language. Also, even though mCRL2 supports minimization modulo strong and weak bisimulation equivalence, it does not output the computed bisimulation, a feature that we have implemented in TMACS.

1.2 Outline

The contributions of this paper are organized as follows. Section 2 presents the implementation of the CCS process language and generation of labeled transition systems as a semantic model of process expressions. Section 3 describes checking the equivalence between two labeled transition systems with respect to behavioural equivalences such as strong weak bisimilarity. It includes implementation details of two algorithms for computing strong bisimulation equivalence, the naive algorithm [5] and the advanced algorithm due to Fernandez [14]. It also entails a description of the saturation technique together with a respective algorithm for saturating a labeled transition system [5]. Next, in Section 4, we

illustrate the application of TMACS for modeling, specification and verification on one classical example in the concurrency theory: the alternating bit protocol [15][16]. Finally, we give some conclusions and some directions for future development of the tool in Section 5.

2 CCS parsing and labeled transition system generation

Given a set of action names, the set of CCS processes is defined by the following BNF grammar:

$$P ::= \emptyset \mid a.P_1 \mid A \mid P_1 + P_2 \mid P_1|P_2 \mid P_1[b/a] \mid P_1\backslash a \tag{1}$$

Two choices were considered for the problem of building a parser in TMACS. The first choice was to build a new parser, which required much more resources and the second choice was to define a grammar and to use a parser generator (compiler-compiler, compiler generator) software to generate the parser source code. In formal language theory, a context-free grammar [18] is a grammar in which every production rule has the form $V \rightarrow w$, where V e is a nonterminal symbol, and w is a string of terminal and/or nonterminal symbols (w can be empty). Obviously, the defined BNF grammar for describing the CCS process is a CFG. Deterministic context-free grammars [18] are grammars that can be recognized by a deterministic pushdown automaton or equivalently, grammars that can be recognized by a LR (left to right) parser [19]. Deterministic context-free grammars are a proper subset of the context-free grammar family [18].

2.1 Generating the parser

ANTLR (ANother Tool for Language Recognition) [20] is a parser generator that was used to generate the parser for the CCS grammar in TMACS. ANTLR uses LL(*) parsing and also allows generating parsers, lexers, tree parsers and combined lexer parsers. It also automatically generates abstract syntax trees [19], by providing operators and rewrite rules to guide the tree construction, which can be further processed with a tree parser, to build an abstract model of the AST using some programming language.

ANTLR was used to generate lexer, parser and a well defined abstract syntax tree which represents a tree representation of the abstract syntactic structure of the parsed sentence. The term abstract is used in a sense that the tree does not represent every single detail that exists in the language syntax, e.g., parentheses are implicit in the tree structure. From a development point of view it is much easier to work with trees than with a list of recognized tokens. One example of an abstract syntax tree is shown in Fig. 1 which is the result of parsing the expression:

$$A = (b.B \mid c.D \ + \ d.D)\backslash \{b, \ c\} \tag{2}$$

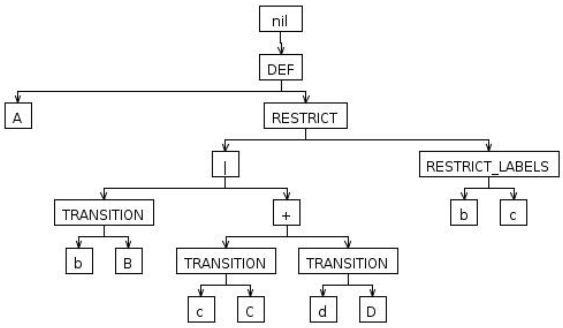


Fig. 1. Example of an abstract syntax tree

2.2 CCS domain model and labeled graph generation

Although working directly with abstract syntax trees and performing all algorithms on them is possible, it causes a limitation in future changes, where even a small change in the grammar and/or in the structure of the generated abstract syntax trees causes a change in the implemented algorithms. Because of this, a specific domain model was built along with a domain builder algorithm, which has corresponding abstractions for all CCS operators, processes and actions. The input of the domain builder algorithm is an abstract syntax tree, and the output is a fully built domain model.

The algorithm for generation of labeled transition system implemented in TMACS is a recursive algorithm which traverses the tree structure of objects in the domain model and performs SOS rule every time it reaches an operation. In this fashion all SOS transformation are performed on the domain and as result a new graph structure is created which represents the labeled transition system which can be easily exported to a file in Aldebaran format.

3 Minimization and comparison of labeled transition systems

Bisimulation equivalence (bisimilarity) [6] is a binary relation between labeled transition systems which associates systems that can simulate each other's behaviour in a stepwise manner, enabling comparison of different transition systems [9]. An alternative perspective is to consider the bisimulation equivalence as a relation between states of a single labeled transition system, providing means for construction of smaller models of the system using the quotient transition system under such a relation [9].

TMACS implements both options: reducing the size of the state space of a given labeled transition system and checking the equivalence of two labeled transition systems, using two behavioral equivalence relations: strong bisimilarity and observational equivalence (weak bisimilarity).

3.1 Minimization of a labeled transition system modulo strong bisimilarity

The process of reducing the size of the state space of a labeled transition system $A = (S, Act, \rightarrow, s, F)$ in TMACS was implemented using an approach which consists of two steps:

1. Computing strong bisimulation equivalence (strong bisimilarity) for the labeled transition system;
2. Minimizing the labeled transition system to its canonical form using the strong bisimilarity obtained in the first step;

Two different methods were used for computing the strong bisimulation equivalence: the so called naive method and a more efficient method due to Fernandez, both of which afterwards serve as minimization procedures.

The naive algorithm for computing strong bisimulation over finite labeled transition systems stems directly from the theory of partially ordered sets and lattices [5] underlying Tarski's classic fixed point theorem [21]. This algorithm has time complexity of $O(mn)$ for a labeled transition system with m transitions and n states. Its implementation in TMACS takes as an input a labeled transition system in Aldebaran format, and generates the corresponding labeled graph as a list of nodes representing the states which use the following data structure:

- $S_p = \{(a, q) \mid p \xrightarrow{a} q,\ p, q \in S,\ a \in Act\}$ - set of pairs (a, q) for state p where a is an outgoing action for p and q is a state reachable from p with action a

The algorithm then computes the strong bisimulation equivalence and outputs it as pairs of bisimilar states $L = \{(p, q) \mid\ p \sim q,\ p, q \in S\}$.

The algorithm due to Fernandez exploits the idea of the relationship between strong bisimulation equivalence and the relational coarsest partition problem [22] solved by Paige and Tarjan in $O(m \log n)$ time [23]. Fernandez's adaptation of the algorithm of Paige-Tarjan has the same complexity as the original one, major difference being that a refinement step is made with only one element of Act in the original one. Our implementation of the algorithm takes as an input a labeled transition system in Aldebaran format, generates a labeled graph and then partitions the labeled graph into its coarsest blocks where each block represents a set of bisimilar states. To define graph states and transitions the following terminology is used represented by suitable data structures:

- $T_a[p] = \{q\}$ - an a-transition from state p to state q
- ${T_a}^{-1}[q] = \{p\}$ - an inverse a-transition from state q to state p
- ${T_a}^{-1}[B] = \cup \left\{{T_a}^{-1}[q], q \in B\right\}$ - inverse transition for block B and action a
- W - set of sets called splitters that are being used to split the partition
- infoB(a, p) - info map for block B, state p and action a

The algorithm of Fernandez outputs the strong bisimulation equivalence relation over *Proc* as a partition $P = \left\{B_i \mid p \sim q,\ \forall p, q \in B_i, i = \overline{1, n}\right\}$ where B_i, $i = \overline{1, n}$, represent its equivalence classes.

Having computed the strong bisimulation equivalence, the next step in the reduction of the state space of the labeled transition system uses the bisimulation equivalence obtained in the first step in order to minimize the labeled graph. This reduction is implemented as follows:

1. All states in a bisimilar equivalence class B_i are merged into one single state $k = \bigcup p_j$, for $p_j \in B_i$;
2. All incoming transitions $r \xrightarrow{a} p_j$, for $p_j \in B_i$, are replaced by transitions $r \xrightarrow{a} k$;
3. All outgoing transitions $p_j \xrightarrow{a} t$, for $p_j \in B_i$, are replaced by transitions $k \xrightarrow{a} t$;
4. The duplicate transitions are not taken into consideration.

The procedure is repeated for all equivalence classes B_i, $i \in \overline{1,n}$.

3.2 Minimization of a labeled transition system modulo weak bisimilarity

The minimization of a labeled transition system modulo weak bisimilarity is reduced to the problem of minimization modulo strong bisimilarity, using a technique called saturation. Saturation first precomputes the weak transition relation and then constructs a new pair of finite processes whose original transitions are replaced by the weak transitions [5].

The algorithm for saturation in TMACS is implemented as follows:

1. For every $p \in S$, the set of all transitions T of a labeled system with m transitions and n states is partitioned in $2n$ sets with $T_{\tau p} = \{(p, \tau, q) \,|\, (p, \tau, q)\}$ and $T_{ap} = \{(p, a, q) \,|\, (p, a, q) \wedge a \neq \tau\}$.
   Then the family of sets $T'_{\tau p}$ is iteratively constructed as:

   $$T^0_{\tau p} = T_{\tau p} \cup \{(p, \tau, p)\},$$
   $$T^i_{\tau p} = T^{i-1}_{\tau p} \cup \left\{(p, \tau, q') \,|\, (\exists q \in S)\,(p, \tau, q) \in T^{i-1}_{\tau p} \wedge (q, \tau, q') \in T_{\tau q}\right\}, \text{and}$$
   $$T'_{\tau p} = T^n_{\tau p}$$

   With this step a reflexive, transitive closure of τ is constructed:

   $$T^*_\tau = \bigcup_{p \in S} T'_{\tau p} = \left\{(p, \tau, q) \,|\, p \left(\xrightarrow{\tau}\right)^* q\right\}$$

2. Next, $T'_s = \bigcup_{p \in S} T'_{ap} = \left\{(p, a, q) \,|\, (\exists q' \in S)\,(p, a, q') \in T \wedge q' \left(\xrightarrow{\tau}\right)^* q\right\}$ is constructed as follows:

   $$T^0_{sp} = T_{sp},$$
   $$T^i_{sp} = T^{i-1}_{sp} \cup \left\{(p, a, q') \,|\, (\exists q \in S)\,(p, a, q) \in T^{i-1}_{sp} \wedge (q, \tau, q') \in T^{i-1}_{\tau q}\right\} \text{and}$$
   $$T'_{sp} = T^{n|Act|}_{sp}$$

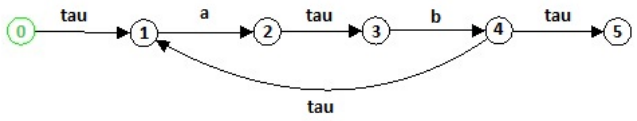
3. $T' = \bigcup_{p \in S} \left(T'_{\tau p} \cup T'_{sp} \right)$ is partitioned again for every $p \in S$, defined by the destination in the transition triple, with $T^*_{\tau q} = \{(p, \tau, q) \,|\, (p, \tau, q) \in T'\}$ and $T_{dq} = \{(p, a, q) \,|\, (p, a, q) \in T' a \neq \tau\}$.
Then the family of sets T^*_{dq} is iteratively constructed as:

$$T^0_{dq} = T_{dq},$$
$$T^i_{dq} = T^{i-1}_{dq} \cup \left\{ (p', a, q) \,|\, (\exists p \in S)\, (p, a, q) \in T^{i-1}_{dq} \wedge (p', \tau, p) \in T^*_{\tau p} \right\}, \text{ and}$$
$$T^*_{dq} = T^{n|Act|}_{dq}$$

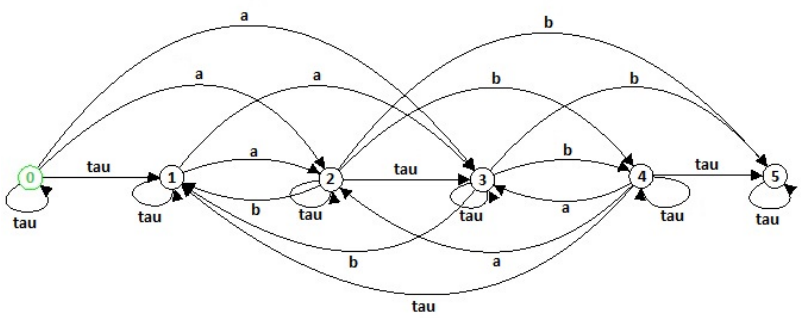
Finally the saturated labeled transition system now is:

$$T^* = \bigcup_{p \in S} \left(T^*_{\tau p} \cup T^*_{dp} \right) =$$
$$= \left(\xrightarrow{\tau} \right)^* \cup \left\{ (p, a, q) \,|\, a \neq \tau \wedge (\exists p', q' \in S)\ p \left(\xrightarrow{\tau} \right)^* p' \xrightarrow{a} q' \left(\xrightarrow{\tau} \right)^* q \right\}$$

An illustration of a saturated labeled transition system before and after applying the saturation technique is given on Fig. 2:



(a) The labeled graph before saturation



(b) The labeled graph after saturation

Fig. 2. Example of a labeled graph before and after applying the saturation technique

Once the saturation algorithm is run and the original labeled transition system is saturated, the computation of weak bisimilarity amounts to computing strong bisimilarity over the saturated system. Afterwards, the process of minimization of the original system is the same as the process of minimization modulo strong bisimilarity applied on the saturated system.

3.3 Comparison of two labeled transition systems modulo strong bisimilarity

The idea for the implementation of the equivalence checking of two labeled transition systems modulo strong bisimilarity was based on the following fact: Two labelled transition systems are (strongly) bisimilar iff their initial states are bisimilar [17]. That means that in order to check whether two labeled transition systems are bisimilar it is enough to check whether their initial states are bisimilar. This can be done using the following approach:

1. The two labeled transition systems are merged into a single transition system
2. An algorithm for computing the strong bisimilarity is applied to the merged system
3. A check is performed to see if the initial states belong to the same bisimulation equivalence class

3.4 Comparison of two labeled transition systems modulo weak bisimilarity

The comparison of two labeled transition systems modulo weak bisimilariy amounts to checking strong bisimilarity over the saturated labeled transition systems [5]. In another words, two labeled transition systems are weakly bisimilar iff their saturated labeled transition systems are strongly bisimilar. Following this fact, we implemented the comparison of two labeled transition systems modulo weak bisimilarity by applying the saturation algorithm over the original labeled graphs in order to obtain their saturated labeled graphs, after which the process of comparison of the saturated labeled transition systems modulo strong bisimilarity was applied as described above.

4 Application

In this chapter we demonstrate that process of formal specification and verification for one classical problem in the concurrency theory: the alternating bit protocol. The alternating bit protocol is a simple yet effective protocol (usually used as a test case), designed to ensure reliable communication through unreliable transmission mediums, and it is used for managing the retransmission of lost messages [5][24].

Modelling and Specification. The representation of the Alternating Bit Protocol consists of a *sender* S, a *receiver* R and two channels *transport* T and *acknowledge* A as shown on Fig. 3.

The only visible transitions in the alternating bit protocol are *deliver* and *accept*, which can occur only sequentially, whereas all others are internal synchronizations. Sender S sends a message which contains the protocol bit, 0 or 1, to a receiver R. The channel from S to R is initialized and there are no messages in transit. There is no direct communication between the sender S and

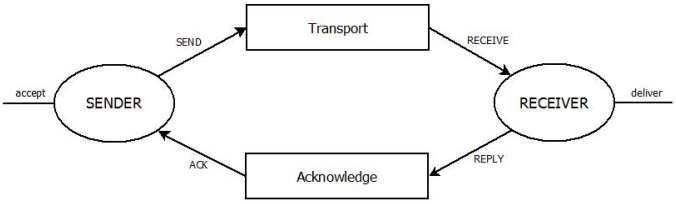


Fig. 3. Alternating Bit Protocol representation

the receiver R, and all messages must travel trough the medium (*transport* and *acknowledge* channel).

The functioning of the alternating bit protocol can be described as [5]:

1. The sender S sends a message repeatedly (with its corresponding bit) until it receives an acknowledgment ($ack0$ or $ack1$) from the receiver R that contains the same protocol bit as the message being sent. This behaviour of the process representing the sender can be described as:

$$S = \overline{send0}.S + ack0.accept.S_1 + ack1.S$$
$$S_1 = \overline{send1}.S_1 + ack1.accept.S + ack0.S_1$$

   The transport channel transmits the message to the receiver, but it may lose the message (lossy channel) or transmit it several times (chatty channel). Therefore, the description of the behaviour of the process representing the transport channel is given with CCS expression as follows:

$$T = send0.\,(T + T_1) + send1.\,(T + T_2)$$
$$T_1 = \overline{receive0}.\,(T + T_1)$$
$$T_2 = \overline{receive1}.\,(T + T_2)$$

2. When the receiver R receives a message, it sends a reply to S which includes the protocol bit of the message received. When the message is received for the first time, the receiver will deliver it for processing, while subsequent messages with the same bit will be simply acknowledged. That yields the following CCS expression for the receiver:

$$R = receive0.\overline{deliver}.R_1 + \overline{reply1}.R + receive1.R$$
$$R_1 = receive1.\overline{deliver}.R + \overline{reply0}.R_1 + receive0.R_1$$

   Again, the acknowledgement channel sends the *ack* to sender, and it can also acknowledge it several times or lose it on the way to the sender. Therefore the ackowledgement channel and its behaviour is described as follows:

$$A = reply0.\,(A + A_1) + reply1.\,(A + A_2)$$
$$A_1 = \overline{ack0}.\,(A + A_1)$$
$$A_2 = \overline{ack1}.\,(A + A_2)$$

3. When S receives an acknowledgment containing the same bit as the message it is currently transmitting, it stops transmitting that message, flips the protocol bit, and repeats the protocol for the next message.[24][13]

Having described the behaviour of the alternating bit protocol components, the CCS process expression describing the behaviour of the protocol as a whole can be obtained as a parallel composition of the processes describing the sender, the transport channel, the receiver and the acknowledgement channel:

$$ABP \stackrel{def}{=} (S|T|R|A) \setminus L, \tag{3}$$

restricted on the set of actions:

$$L = (send0, send1, receive0, receive1, reply0, reply1, ack0, ack1)$$

This CCS expression represents the implementation of the alternating bit protocol which details the proposed means for achieving the desired high-level behaviour the alternating bit protocol should exhibit. This desired high-level behaviour is that the alternating bit protocol should act as a simple buffer, therefore its CCS specification is defined as follows:

$$\begin{array}{l} Buf = accept.Buf' \\ Buf' = \overline{deliver}.Buf \end{array} \tag{4}$$

Verification. In order to verify the alternating bit protocol, we need to prove that the implementation ABP meets the specification Buf with respect to some behavioural equivalence. We shall show that an observational equivalence between Buf and ABP can be found, i.e. that $ABP \approx Imp$. For that purpose, first we use TMACS to obtain the labeled graphs corresponding to the CCS representations of Buf and ABP, and afterwards we perform a comparison of the labeled transition systems modulo weak bisimilarity which yields a positive answer about the existance of weak bisimulation equivalence between Buf and ABP.

The weak bisimulation equivalence obtained by running any of the two bisimulation algorithms implemented in TMACS over the saturated labeled transition systems is given in Table 1:

5 Conclusions and Future Work

In this paper we presented TMACS, a tool for modeling, manipulation and analysis of concurrent systems. It includes recognizing CCS expressions, building labeled transition system graphs and checking equivalence between two labeled graphs with respect to strong and weak bisimilarity. The tool is simple, but yet functional enough to perform specification and verification of concurrent systems described as CCS expressions and/or labeled transition systems. TMACS is already successfully used for the needs of the Official Gazette of the Republic of

ABP implementation states	ABP specification states
$(S\|T\|R\|A)\setminus L$	
$(S\|\,(T+T1)\,\|R\|A)\setminus L$	
$(S\|\,(T+T1)\,\|R\|\,(A+A2))\setminus L$	
$(S\|T\|R\|\,(A+A2))\setminus L$	
$\big(S\|\,(T+T1)\,\|\overline{deliver}.R1\|A\big)\setminus L$	
$\big(S\|\,(T+T1)\,\|\overline{deliver}.R1\|\,(A+A2)\big)\setminus L$	
$(S1\|\,(T+T1)\,\|R1\|\,(A+A1))\setminus L$	
$\big(S1\|\,(T+T2)\,\|\overline{deliver}.R\|\,(A+A1)\big)\setminus L$	
$(S1\|\,(T+T2)\,\|R1\|\,(A+A1))\setminus L$	
$(S\|\,(T+T2)\,\|R\|\,(A+A2))\setminus L$	Buf
$(accept.S1\|\,(T+T1)\,\|R1\|\,(A+A1))\setminus L$	
$(S\|\,(T+T1)\,\|R1\|\,(A+A1))\setminus L$	
$(S\|\,(T+T1)\,\|R1\|\,(A+A2))\setminus L$	
$(S\|\,(T+T1)\,\|R1\|A)\setminus L$	
$(S1\|\,(T+T2)\,\|R\|\,(A+A2))\setminus L$	
$(accept.S\|\,(T+T2)\,\|R\|\,(A+A2))\setminus L$	
$(S1\|\,(T+T2)\,\|R\|\,(A+A1))\setminus L$	Buf'

Table 1. Verification of the alternating bit protocol using weak bisimilarity

Macedonia, a fact which indicates that the tool can be quite useful for practical as well as experimental applications.

Future work can include optimization of the tool's response times for very large labeled transition system graphs, as well as implementation of prunung strategy for infinite labeled transition system graphs. It would be also usefull if the labeled graph is better visualized where every SOS rule is shown, with its source and sink nodes showing the CCS expression for each of the nodes. Furthermore, the minimization and comparison functionality for labeled transition systems which at the moment includes only strong and weak bisimilarity can be extended to include other behavioural equivalences and preorders as well. Another direction for future development can be to extend the tool to support not only CCS expressions and process semantics in general, but also other formal verification approaches as well.

References

1. R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, vol. 92, Springer-Verlag, 1980
2. R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989
3. J.A. Bergstra, A. Ponse, S.A. Smolka, *Handbook of Process Algebra*, Elsevier Science B.V., 2001
4. R.M. Keller, *Formal Verification of Parallel Programs*, Communications of the ACM, 19(7), pp. 371-384, 1976
5. L. Aceto, A. Ingolfsdottir, K. G. Larsen, J. Srba, *Reactive Systems - Modeling, Specification and Verification*, Cambridge University Press, 2007

6. D.M.R. Park, *Concurrency and Automata on Infinite Sequences*, In: Proceedings of the 5th G.I.Conference in Theoretical Computer Science, Lecture Notes in Computer Science, vol. 104, Springer-Verlag, pp. 167-183, 1981
7. A.W. Roscoe, *Understanding Concurrent Systems*, Texts in Computer Science, Springer-Verlag, 2010
8. R. Milner, *Operational and Algebraic Semantics of Concurrency Processes*, In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science, pp. 1201-1242, Elsevier and MIT Press, 1990
9. C. Baier, J.-P. Katoen, *Principles of Model Checking*, The MIT Press, 2008
10. J.C. Fernandez, *Aldebaran: User's Manual*, Technical Report, LGI-IMAG Grenoble, 1988
11. F. Moller, Perdita Stevens, *Edinburgh Concurrency Workbench User Manual (Version 7.1)*, 1999, Available from `http://homepages.inf.ed.ac.uk/perdita/cwb/`
12. M. van Weerdenburg et al., *mCRL2 User's Manual*, 2008, Available from `http://www.mcrl2.org`
13. M.Alexander, W.Gardner, *Process Algebra for Parallel and Distributed Processing*, Chapman&Hall/CRC Press, Computational Science Series, 2009
14. J.C. Fernandez, *An Implementation of an Efficient Algorithm for Bisimulation Equivalence*, Science of Computer Programming, vol. 13, pages 219-236, 1989/1990
15. W.C. Lynch, *Computer systems: Reliable Full-duplex File Transmission over Half-duplex Telephone Line*, Communications of the ACM, vol. 11, no. 6, pp. 407410, 1968
16. K.A. Bartlett, R.A. Scantlebury, P.T. Wilkinson, *A note on Reliable Full-duplex Transmission over Half-duplex Links*, Communications of the ACM, vol. 12, pp. 260261, 1969
17. J.F. Groote, M. Reniers, *Modelling and Analysis of Communicating Systems,* Technical University of Eindhoven, rev. 1478, 2009
18. N. Chomsky, *Three Models for the Description of Language*, IEEE Transactions on Information Theory, 1956
19. A.V. Aho, R. Sethi, J.D. Ullman, *Compilers: Principles, Techniques, and Tools* Addison Wesley, 2006
20. T. Parr, *The Definitive ANTLR Reference - Building Domain-Specific Languages*, The Pragmatic Bookshelf, 2007
21. A. Tarski, *A Lattice-theoretical Fixpoint Theorem and its Applications*, Pacific Journal of Mathematics 5:2, pp. 285309, 1955
22. P. Kanellakis, S.A. Smolka, *CCS expressions, finite state processes and three problems of equivalence*, In Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing, pp. 228-240, ACM Press, 1983
23. R. Paige, R. Tarjan, *Three Partition Refinement Algorithms,* SIAM J. Comput. 16 (6), 1987
24. Seth Kulick, *Process Algebra, CCS, and Bisimulation Decidability*, University of Pennsylvania, pages 8-10, 1994