Design Verification of a Microprocessor Using Branching Time Regular Temporal Logic*

Kiyoharu HAMAGUCHI[†], Hiromi HIRAISHI[‡] and Shuzo YAJIMA[†]

[†] Department of Information Science, Faculty of Engineering,
Kyoto University, Kyoto, 606, JAPAN.
(E-mail: hama@kuis.kyoto-u.ac.jp)

[‡] Department of Information & Communication Sciences,
Kyoto Sangyo University, Kita-ku, Kyoto, 603, JAPAN.

Abstract. This paper reports about design verification of a real microprocessor using a symbolic model checking algorithm for a variant of branching time temporal logics, i.e. branching time regular temporal logic (BRTL).

The 8-bit microprocessor which was verified is KUE-CHIP2, which is an LSI chip developed for educational purpose. The KUE-CHIP2 contains approximately 1,600 gates and 68 flip-flops and supports 19 kinds of instructions, many of which can specify absolute/index addressing. The language for specification is BRTL, which is an enhancement of computation tree logic (CTL). BRTL has automaton connectives as temporal operators. Some examples of the specification in BRTL are shown.

The adopted verification technique is symbolic model checking using binary decision diagram (BDD). Some of the properties such as conditions for buses or behaviors for the instructions were checked. The verifier succeeded in finding a bug. The total verification time for all the instruction was approximately 10 hours.

1 Introduction

In order to verify correctness of finite state machines automatically, various methods have been widely studied. Among them, the model checking approach based on computation tree logic (CTL) has been shown to be one of the most practical and effective methods. The model checking method, however, suffered from combinatorial explosion which can be observed in construction of state transition graphs or Kripke structures from sequential circuits.

On the other hand, as a new data structure to represent and handle logic functions, binary decision diagram (BDD) was formulated by R.E. Bryant[1]. Through the development of efficient BDD manipulators[2, 3], it has been known that BDD can relax effectively the exponential blowups which occur in manipulation of logic functions

The key idea of symbolic model checking proposed by Burch et al. [4] is to express sequential circuits by logic functions and to perform model checking

* This research was partially supported by Japan-USA Cooperative Research sponsored by JSPS and NSF. The first author was partially supported by Grant-in-Aid for Scientific Research (B)(1) under project number 04555079.

through logic function manipulation. Using BDD for manipulating logic functions, the symbolic model checker for CTL succeeded in verifying a pipelined ALU, which has 10^{20} states in the form of state transition diagrams[4]. The circuit size which can be handled has increased by several works [5, 6]. Recently, by introducing abstraction through homomorphism to reduce a register file composed of 64 registers to 3 registers, the pipelined ALU with 10^{1300} states was successfully verified[7].

The aim of this paper is to report about design verification of a real microprocessor based on the symbolic model checking method.

An 8-bit microprocessor which was verified is KUE-CHIP2 (Kyoto University Educational Chip 2), which is an LSI chip developed for educational purpose. The KUE-CHIP2 which is composed of approximately 1,600 gates and 68 flipflops is more complex in some points than the above pipelined ALU. For example, the processor supports 19 kinds of instructions, many of which specify absolute/index addressing modes and the phase length of the instructions varies in the range from 3 to 5.

The specification is described by using branching time regular temporal logic (BRTL)[8], which is an improvement of CTL in terms of expressive power. BRTL uses a restricted ω automaton of Büchi type to express temporal properties. Although the expressive power is smaller than extended CTL (ECTL)[9], the restriction to ω automata gives less computation cost in the symbolic model checking. For example, complex nested fixed point operations can be avoided.

The reason BRTL instead of CTL or μ calculus([10] etc.) was used, is that ω automaton is straightforward to express temporal properties especially when we handle the structure corresponding to *while*. Furthermore it is easier to understand intuitively than greatest or least fixed point operators of μ calculus. In a later section, we show some examples of how to describe specifications for the microprocessor in BRTL.

Using a symbolic model checking algorithm for BRTL, some of the properties such as conditions for buses or behaviors for the instructions were checked and bugs were found. (In the verification, the abstraction method was not utilized.) The experimental results suggest that the verification of the whole chip (excluding the internal memory) can be performed in reasonable time.

This paper is organized as follows: Section 2 overviews design verification using BRTL. Section 3 explains the algorithm of symbolic model checking through an example. Section 4 describes some examples of specification for the KUE-CHIP2 and reports experimental results. Section 5 concludes this paper. The accurate definition of BRTL and the detail of the algorithm are shown in Appendix.

2 Overview of Design Verification

This section overviews how a given sequential machine is verified for a specification described in BRTL.

When we consider a tuple of input values and output values, i.e. an input/output tuple, the behavior of a sequential machine corresponds to a set of sequences composed of the input/output tuples. The model checking algorithm

checks whether the property which a given BRTL formula specifies is satisfied by the set of the sequences of the input/output tuples.

In the following, the values allowed to use as inputs and outputs are restricted to 1 or 0, without loss of generality. Boolean operators $\vee$, $\neg$, $\wedge$, $\equiv$ and $\Rightarrow$ mean disjunction, negation, conjunction, equivalence and implication as usual. *true* and *false* represent tautology and inconsistency respectively. $+$, $-$, $\cdot$ are sometimes used instead of $\vee$, $\neg$, $\wedge$ respectively.

(1) Transformation to Kripke structure

We assume designs are given as sequential machines. Since the semantics of BRTL formulas is determined over Kripke structures, A Kripke structure has to be obtained so that the structure reflects all the behaviors of the sequential machine.

The sequential machine of Moore type shown in Figure 1(a) is transformed to the Kripke structure shown in Figure 1(b). The part in the dashed line in (a) is transformed to the vertex s_1 in (b). The initial vertices corresponding to the initial state of the machine is pointed by $\Leftarrow$

Fig. 1. Transformation to Kripke structure

Fig. 2. Automata Connectives

(2) Description of specification

We see how the following specification is described in BRTL.

"It always holds that, when the input x becomes 1, then the output changes its value at the next time"

The property is expressed as follows:

$$\forall \text{Always}(x \Rightarrow ((z \Rightarrow \forall \text{Next}(\neg z)) \wedge (\neg z \Rightarrow \forall \text{Next}(z))))$$

Here $\forall$ means "over all paths on the Kripke structure". $\text{Always}(f)$ and $\text{Next}(f)$ mean " f always holds" and " f holds at the next time" respectively. The temporal properties such as Always and Next are described by finite automata shown in Figure 2.

The automata have propositional logic formulas (more generally, BRTL formulas) as their transition conditions. The finite automata in BRTL accept or reject infinite sequences of vectors composed of 0 or 1. The acceptance condition is Büchi type, that is, an automata accepts an infinite sequence if and only if it hits some accepting state infinitely often. The automata satisfying the following conditions are allowed to use in BRTL.

- It has a unique initial state.
- The transition is deterministic and completely specified.
- There exists no path from an arbitrary rejecting state q_r to q_r via some accepting state.

For automata A , A_1 and A_2 , $\forall A$, $\exists A$, $\forall(A_1 \vee A_2)$, $\forall \neg A$ and so on can be described in BRTL.

(3) Verification

Whether a BRTL formula holds or not is determined at each vertex on a Kripke structure. A formula $\forall A$ ($\exists A$) holds at a vertex if and only if all (some) paths on the Kripke structure starting from the vertex are accepted.

For example, in Figure 1(b), $\forall \text{Next}(z)$ holds at s_2 and $g \stackrel{\text{def}}{=} x \Rightarrow ((z \Rightarrow \forall \text{Next}(\neg z)) \wedge (\neg z \Rightarrow \forall \text{Next}(z)))$ holds at s_0, s_1, s_2 and s_3 .

The given specification is satisfied if and only if the BRTL formula holds at every initial vertex of the Kripke structure.

3 Outline of Symbolic Model Checking

The detail of the algorithm of symbolic model checking is shown in Appendix.

The following notations are used. f is a logic function. $\exists x.f(x, y) \stackrel{\text{def}}{=} f(0, y) \vee f(1, y)$. For $\mathbf{x} = x_1, x_2, \dots, x_n$, $\exists \mathbf{x}.f(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \exists x_1.(\exists x_2. \dots (\exists x_n.f(\mathbf{x}, \mathbf{y})) \dots)$

Furthermore, for $B = \{0, 1\}$, a subset $X \subseteq B^n$ is represented by a logic function with n variables f_X such that f_X satisfies the following condition:

$$x \in X \Leftrightarrow f_X(x) = 1$$

When a sequential machine is given as a design, a unique code word is assigned to each state of the machine and transition functions and output functions of the machine are used as inputs to the model checking algorithm.

Let $x_i (1 \leq i \leq l)$ and $y_j (1 \leq j \leq m)$ be input variables and state variables over B respectively. Vectors of variables are represented like $\mathbf{x} = x_1, x_2, \dots, x_l$.

- Transition functions: $f_j : B^l \times B^m \rightarrow B (1 \leq j \leq m)$
- Output functions: $z_k : B^m \rightarrow B (1 \leq k \leq n)$ (Moore type)
 $z_k : B^l \times B^m \rightarrow B (1 \leq k \leq n)$ (Mealy type)

Let $g_{init}(\mathbf{y})$ be a logic function representing the set of initial states of the machine.

At first, a logic function representing the set of edges of the Kripke structure obtained from the transition functions.

(1) Transformation to Kripke structure

The set of vertices of the Kripke structure comes to be the product space $B^l \times B^m$ composed from the domain of inputs and the domain of state variables. Let $\mathbf{s} \stackrel{\text{def}}{=} \mathbf{x} \# \mathbf{y}$. $\#$ expresses the concatenation of $\mathbf{x}$ and $\mathbf{y}$. For $\mathbf{x}$, new variables $\mathbf{x}' = x'_1, x'_2, \dots, x'_k$ are introduced. Similarly, $\mathbf{y}'$ are introduced. Let $\mathbf{s}' = \mathbf{x}' \# \mathbf{y}'$. $\mathbf{s}$ and $\mathbf{s}'$ are used to represent starting points and ending points of edges. The logic function f_S representing the set of edges of the Kripke structure S can be constructed as follows:

$$f_S(\mathbf{s}, \mathbf{s}') = \bigwedge_{0 \leq j \leq m} (y'_j \equiv f_j(\mathbf{x}, \mathbf{y}))$$

The logic function f_{init} representing the initial vertices of the Kripke structure is defined by $f_{init}(\mathbf{s}) = g_{init}(\mathbf{y})$. The logic functions representing the set of vertices where the value of x_i or z_k is 1, come to be $f_{x_i}(\mathbf{s}) = x_i$, $f_{z_k}(\mathbf{s}) = z_k(\mathbf{y})$ (or $z_k(\mathbf{x}, \mathbf{y})$), respectively.

(2) Algorithm

Firstly, for each subformula ψ_i contained in the given BRTL formula ψ , the logic function satisfying the following conditions are calculated in bottom up manner.

$$f_{\psi_i}(\mathbf{s}) = 1 \Leftrightarrow \psi_i \text{ holds at } \mathbf{s}$$

Secondly, $f_{init}(\mathbf{s}) \Rightarrow f_{\psi}(\mathbf{s})$ is obtained. The algorithm outputs 'yes' if and only if this logic function is *true*. The calculation of $f_{\psi}(\mathbf{s})$ is done by the algorithm shown in Appendix.

It is necessary to perform manipulation of logic functions efficiently, when we implement the above algorithm. Shared binary decision diagram (SBDD)[2] is an improvement of the original BDD[1], which shares all possible subgraphs among multiple functions. By using BDD, logical operations, substitution to logical variables and equivalence checking of two functions can be performed efficiently and many functions are represented compactly. The implemented system uses SBDD.

For the purpose of efficient model checking, the whole transition relation functions f_S are not generated. Each of transition functions is handled separately using the method shown in [5, 6].

4 Verification Results

4.1 Specification

In this section, examples of specification description are shown. The verified microprocessor is KUE-CHIP2. The KUE-CHIP2 is an 8-bit microprocessor based on CMOS technology and the second version of KUE-CHIP[11] which was developed for educational purpose. The microprocessor contains approximately 1,600 gates (exclusive of the internal memory) and 68 flipflops and supports 19 kinds of instructions, many of which can specify absolute/index addressing modes. The microprocessor does not contain a pipeline structure nor a multiplier. The architecture is shown in Figure 3.

Fig. 3. The Architecture of KUE-CHIP2

The microprocessor has three kinds of operation modes: normal mode, single instruction mode and single phase mode. Under the single instruction (or phase) mode, only an instruction (or a phase) is performed by a start pulse.

The KUE-CHIP2 has an external memory and an internal memory. Since direct manipulation of 8 bit by 512 word memory is impossible even with BDD, inputs (or outputs) of the memory are modeled as outputs (or inputs) of the microprocessor. Specifications have to be described so that the contents of the memory are not accessed.

Phase signals

The following formula means that "if 'op' is always 1, then the phase number of each instruction is 3, 4 or 5, and the phase signals become 1 correctly". 'op'

is a signal which indicates that the microprocessor is in operation. 'ph0' is an abbreviation of $ph0 \cdot \overline{ph1} \cdot \overline{ph2} \cdot \overline{ph3} \cdot \overline{ph4}$ and so on. $ph\ i$ is the signal representing the phase i at which the processor is operating.

$$\forall Always(\forall(Always(op) \Rightarrow A))$$

A is shown in Figure 4. The edges to the dead state are abbreviated.

Conditions for Buses

The condition for the bus shown in Figure 7 can be described as follows:

$$\forall Always(a\overline{b}\overline{c} + \overline{a}b\overline{c} + \overline{a}\overline{b}c)$$

The formula claims that one and only one control signal among the signals of the tristate buffers is 1. *Always* is shown in Figure 2.

Behaviors for Instructions

The behavior for each instruction is verified by checking the behavior for each bit and for each phase. For example, the behavior of the instruction IN is informally described by Figure 5. The following formula specifies the behavior for $(ibuf) \rightarrow acc$.

$$\forall Always(\forall(Always(op) \Rightarrow IN))$$

IN is shown in Figure 6. Instruction IN fetches data from the input buffer and put it to the accumulator. 'ibuf0' and 'acc0' represents the 0th bit of input data and the 0th bit of the accumulator (ACC) respectively. Similar description for every bit is used as specification. 'op' is the signal representing that the processor is operating. ph_0 means phase 0 as shown in the above. 'opin' means the code for IN. 'opin' is a logic formula $\overline{mem7} \cdot \overline{mem6} \cdot \overline{mem5} \cdot \overline{mem4} \cdot mem3$, where $mem\ i$ represents the i -th bit of the outputs of the memory.

When the processor was verified, different descriptions were given for efficiency. Since the first two edges from the initial state of IN are the same for each bit, they can be shared. The automaton IN was split and the same property was expressed by nesting two automata.

Fig. 4. Automaton connective A

Phase 0	Phase 1	Phase 2	Phase 3
(pc) $\rightarrow$ mar pc++	(mem) $\rightarrow$ ir	(ibuf) $\rightarrow$ acc 0 $\rightarrow$ ibuf_re	0 $\rightarrow$ ifc

Fig. 5. Behavior of IN (phase 2 and 3)

Fig. 6. Automaton connective *IN*

4.2 Experimental Results

A symbolic model checker for BRTL was implemented. The verifier was written in language C and runs on a SPARCstation 2. This program utilizes the Boolean function manipulator developed by Minato et.al. [2].

The transition functions and the output functions were obtained directly from the design data made on SOLO, a CAD tool developed by ES2 (European Silicon Structure).

Each flip-flop was expanded to a transition function using a next state variable. The internal memory (8 bit $\times$ 512 words) included in KUE-CHIP2 was detached in the translation, that is, it was treated as an external memory. The bus shown in Figure 7 was translated to the following formula.

$$bus = (a \Rightarrow d_a) \wedge (b \Rightarrow d_b) \wedge (c \Rightarrow d_c)$$

Fig. 7. Bus

In order to describe the ω finite automata, we implemented a simple language which has structures of 'while' or 'if' and can use macro definitions.

The number of nodes of SBDD was limited to 500,000 (11 M byte).

The conditions for all the control signals for the buses were checked. The verifier found that a bus floats under some condition, and the error was debugged.

Table 1 shows the experimental result of verifying some of the properties of the microprocessor. 'bus' shows the result for the conditions for the buses. 'phloop' shows the result for $\forall Always(\forall(Always(op) \Rightarrow A))$. Since the behaviors at phase 0 and 1 are common among all the instructions, they were checked separately without specifying any instruction. The results for the phases 0 and 1 are shown in 'ph0' and 'ph1' respectively. The result for each instruction does not consider the behaviors for phase 0 and 1. The given specifications include the conditions where the registers that should hold the values, for example, the index register in Figure 5, do not change their values.

In order to simplify the verification, some of the inputs to the processor were fixed to 0 or 1.

'ADC' in Table 1 is the instruction with the address modification by the index register. The behavior for the instruction is most complex.

The KUE-CHIP2 has about 170 instructions, if addressing modes and destination registers are distinguished. The behavior of all the instructions under the normal mode of the KUE-CHIP2 was verified. The total verification time was approximately 10 hours. The size of the files containing the specifications was approximately 2 Mbyte.

Table 1. Experimental Results for KUE-CHIP2

	time	# PH	
bus	12.8	—	Bus conditions
phloop	16.2	—	Phase signals
ph0	21.3	—	Behavior for phase 0
ph1	48.6	—	Behavior for phase 1
RCF	81.8	3	Reset Carry Flag
IN	114.8	4	INput to ACC
OUT	136.8	4	OUTput from ACC
LD†	234.7	5	LoaD from memory to ACC
ST†	242.3	5	STore from ACC to memory
ADC†	309.8	5	ADDition with Carry

time: CPU time (sec)

PH: the number of phases for each instruction

†: modified by the index register

5 Conclusion

In this paper, for an 8-bit microprocessor KUE-CHIP2, how to describe specification in a variant of branching time temporal logics, i.e. BRTL, was shown. The specifications were checked by using a symbolic model checking method. The experimental results show that the symbolic model checking method is suitable to the microprocessor design and that the verification of the whole chip (excluding the internal memory) can be done in reasonable time.

Although the KUE-CHIP2 has only 8-bit arithmetic operations excluding multiplication, the 16 or 32-bit multiplier increases the difficulties of verification drastically. In order to verify larger microprocessors, incorporation of the technique shown in [12] and so on has to be investigated.

The descriptions of specification in Section 3 and 4 are given for each phase. It would be better to describe the specification as a relation between the condition at the first phase and that at the last phase for each instruction as shown in [4]. In order to verify such descriptions, the part of the internal memory has to be handled as a sequential circuit. Abstraction mechanism seems powerful to handle large memories. We would like to introduce the technique and estimate its effectiveness in the near future.

Acknowledgment

The authors would like to express sincere appreciation to Prof. Hiroto Yasuura and Mr. Hiroyuki Kanbara who provided the design data of KUE-CHIP2. They would also like to thank Mr. Shin-ichi Minato who offered the Boolean function manipulation subroutines using SBDD representation, and Mr. Fuminobu Yatsuboshi who performed the verification of all the KUE-CHIP2 instructions.

References

1. R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677-691, August 1986.
2. Shinichi Minato, Nagisa Ishiura, and Shuzo Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. In *Proceedings of 27th Design Automation Conference*, pages 52-57, 1990.
3. K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of 27th Design Automation Conference*, pages 40-45, 1990.
4. J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *Proceedings of 27th Design Automation Conference*, pages 46-51, 1990.
5. J. R. Burch, E. M. Clarke, and D. E. Long. Representing Circuits More Efficiently in Symbolic Model Checking. In *Proceedings of 28th Design Automation Conference*, pages 403-407, June 1991.
6. H. Hiraishi, K. Hamaguchi, H. Ochi, and S. Yajima. Vectorized Symbolic Model Checking of Computation Tree Logic. In *Workshop on Computer-Aided Verification*, pages 279-290, July 1991.
7. E. M. Clarke and O. Grumberg and D. E. Long. Model Checking and Abstraction. Technical report, Carnegie Mellon University, 1991. manuscript.

8. K. Hamaguchi, H. Hiraishi, and S. Yajima. Branching Time Regular Temporal Logic for Model Checking with Linear Time Complexity. In *Workshop on Computer-Aided Verification*, June 1990.
9. E. M. Clarke, O. Grumberg, and R. P. Kurshan. A synthesis of two approaches for verifying finite state concurrent systems. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*. Springer Verlag, July 1989.
10. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of 5th IEEE Symposium on Logic in Computer Science*, June 1990.
11. H. Kanbara. KUE-CHIP: A Microprocessor for education of Computer Architecture and LSI design. In *Proc. of IEEE ASIC Seminar & Exhibit*, 1990.
12. J. R. Burch. Using BDDs to verify multipliers. In *Proceedings of 28th Design Automation Conference*, pages 408-412, jun 1991.
13. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

Appendix

Syntax and Semantics of BRTL

BRTL uses the following dfa-1 as temporal operators.

Definition 1. A *deterministic ω finite automaton type 1 (dfa-1)* $A = (Q, P, Br, q_0, F)$ is defined as follows:

Q is a set of finite states, $P = \{p_1, \dots, p_n\}$ is a set of propositional variables. $Br : Q \times Q \rightarrow BF$ (BF is a set of propositional formulas composed from P) is a partial function which assigns a propositional formula to a pair of two states. Let $Br(q, Q)$ be $\{f \mid \exists q'. Br(q, q') = f, q' \in Q\}$, for $q \in Q$.

Br satisfies the following conditions: "For all $f_1, f_2 \in Br(q, Q)$ such that $f_1 \neq f_2$, $f_1 \wedge f_2 = \text{false}$ " and " $\bigvee_{f \in Br(q, Q)} f = \text{true}$." q_0 is an initial state and F is a set of accepting states. The elements of $Q - F$ are called rejecting states. Furthermore A satisfies the following condition.

"There exists no path from an arbitrary rejecting state q_r to q_r via some accepting state"

A accepts or rejects infinite sequences composed of the elements of $\Delta = 2^P$. The transition function $\delta : Q \times \Delta \rightarrow Q$ is defined as $\delta(q, v) = q' \Leftrightarrow Br(q, q')(v) = T$ for $q, q' \in Q, v \in \Delta$. $Inf(\sigma)$ represents of the set of states which A goes through infinitely often when σ composed of the elements of Δ is given to A as an input. The language which A accepts is $\{\sigma \mid Inf(\sigma) \cap F \text{ is not empty}\}$. $\square$

Proposition 2. [8] For a dfa-1 $A = (Q, P, Br, \delta, q_0, F)$, the dfa-1 $\bar{A}$ which accepts the complementary set of A can be obtained by exchanging the accepting states with the rejecting states. Furthermore, the dfa-1 $(A_1 | A_2)$ which accepts the union of the sets accepted by A_1 and A_2 can be constructed. In other words, dfa-1's are closed under Boolean operations. $\square$

Definition 3. Syntax

BRTL formula : BRF represents the set of BRTL formulas.

$p \in AP$, $\psi, \phi \in BRF$. If A is an automaton connective, then $p, (\neg\psi), (\psi \vee \phi), (\exists A) \in BRF$.

Automaton connective

Let AP be a set of propositional variables and A_i ($i = 1, \dots, m$) be dfa-1's. We assume that $\{\psi_1, \psi_2, \dots, \psi_n\} \subseteq BRF$. $A(\psi_1, \psi_2, \dots, \psi_n)$ is obtained by replacing each propositional variable $p_1, p_2, \dots, p_n \in AP$ in A with $\psi_1, \psi_2, \dots, \psi_n$ simultaneously. The value of $Br(q, q')$ changes accordingly. Any description which is obtained by finite number of applications of $\neg$ (unary operator) and $\vee$ (binary operator) to $A_i(\psi_1, \psi_2, \dots, \psi_n)$, is called an automaton connective. $\square$

Proposition 2 shows that $\exists \neg A$ and $\exists(A_1 \vee A_2)$ can be expressed by $\exists \bar{A}$ and $\exists(A_1 | A_2)$ respectively.

The semantics of BRTL formulas is defined for Kripke structures. Let AP be the set of atomic propositions $p_1, p_2, \dots, p_n$.

Definition 4. $S = \langle \Sigma, I, R, \Sigma_0 \rangle$ is called Kripke structure. Σ is a set of vertices. $I : \Sigma \rightarrow 2^{AP}$ assigns the truth value of each atomic proposition to each vertex. $R \subseteq \Sigma \times \Sigma$ is a set of directed edges over Σ and Σ_0 is a set of initial states. $\square$

Definition 5. Semantics

$S, s \models \psi$ means that a BRTL formula ψ holds at the vertex $s \in \Sigma$ (or ψ is true at s). We assume $p \in AP$, and $\psi, \phi, \psi_1, \psi_2, \dots, \psi_n$ are BRTL formulas. A is a dfa-1.

- $S, s \models p \Leftrightarrow p \in I(s)$
- $S, s \models (\psi \vee \phi) \Leftrightarrow S, s \models \psi$ or $S, s \models \phi$
- $S, s \models (\neg\psi) \Leftrightarrow S, s \not\models \psi$
- $S, s \models (\exists A(\psi_1, \psi_2, \dots, \psi_n)) \Leftrightarrow$ There exist an infinite sequence $s_0 s_1 s_2 \dots$ starting from s on the Kripke structure S and an infinite sequence of states of A such that $S, s_i \models Br(q_i, q_{i+1})$ ($i = 0, 1, 2, \dots$) and at least one state which appears infinitely often along $q_0, q_1, \dots$ is an accepting state of A .
- $S, s \models (\exists \neg A) \Leftrightarrow S, s \models (\exists \bar{A})$
- $S, s \models (\exists(A_1 \vee A_2)) \Leftrightarrow S, s \models (\exists(A_1 | A_2))$

When $S, s \models \psi$ holds for all $s \in \Sigma_0$, then we describe $S \models \psi$. $\square$

$\forall$ is defined as follows: $\forall A \stackrel{\text{def}}{=} \neg \exists \neg A$.

Symbolic model checking

Algorithm 1 Symbolic model checking algorithm

- Input: f_S, f_{init}, f_{x_i} and f_{z_k} , for a BRTL formula ψ and a Kripke structure S .
- Output: If $S \models \psi$ then yes. Otherwise no.

– Method:

1. By Proposition 2, ‘ $\neg$ ’ and ‘ $\vee$ ’ are removed from each automaton connective A_j .
2. For each subformula ϕ_i in ψ , $f_{\phi_i}(s)$ which represents the set of vertices such that $S, s \models \phi_i$ is obtained in bottom up manner.
 Let f_{h_1}, f_{h_2}, f_h represent the set of vertices s such that $S, s \models h_1, S, s \models h_2, S, s \models h$ respectively.
 - (a) Case: ϕ_i is an atomic proposition.
 f_{x_i} and f_{z_k} are given.
 - (b) Case: $\phi_i = h_1 \vee h_2$ or $\neg h$.
 $f_{h_1 \vee h_2} = f_{h_1} \vee f_{h_2}, f_{\neg h} = \neg f_h$.
 - (c) Case: $\phi_i = \exists A_j$
 Regarding A_j as a directed graph (V, E) , strongly connected components (SCC’s) are calculated. See the definition and the algorithm for SCC’s in [13, Chap. 5.5]. (The definition claims that an SCC may consists of only one vertex without a selfloop. Let q_0 be the initial state of A_j . Then $f_{\phi_i} = \text{fixpoint}(q_0)$.
 The procedure `fixpoint` and `label_accept` and `label_all` are shown in Figure 8, Figure 9 and Figure 10. `label_function(q, f)` labels f to the state q of the automaton A_j . `labeled_function(q)` returns the logic function labeled to q . $\bigvee_{i \in S} f_i$ (f_i is a logic function) is assumed to be *false*, if S is an empty set. BF is shown in Section 3.
3. If $(f_{init}(s) \Rightarrow f_\eta(s)) = \text{true}$, yes. Otherwise no. □

```

procedure fixpoint( $q$ )
/* ( $V_q, E_q$ ) is an SCC containing  $q$ . */
  begin
1.   for all  $q' \in V_q$  do label_function( $q'$ , false);
2.   if  $q$  is an accepting state then label_accept( $q$ );
3.   for all  $q' \in V_q$  do
      begin
4.        $f = \text{labeled\_function}(q') \vee$ 
           $\bigvee_{(q', q'') \in E, q'' \notin V_q} (BF(\text{fixpoint}(q'')) \wedge f_{Br(q', q'')})$ ;
5.       labeled_function( $q'$ ,  $f$ );
      end
6.   label_all( $q$ );
7.   return (labeled_function( $q$ ));
  end

```

Fig. 8. Procedure `fixpoint`

```

procedure label.accept( $q$ )
  /* ( $V_q, E_q$ ) is an SCC containing  $q$ . */
  begin
1.   for all  $q' \in V_q$  do label_function( $q', true$ ); end
2.   repeat
3.     for all  $q' \in V_q$  do
4.       begin
5.          $f = \text{labeled\_function}(q') \wedge$ 
            $\bigvee_{(q', q'') \in E_q} ( \text{BF}(\text{labeled\_function}(q'')) \wedge f_{Br(q', q'')} );$ 
6.         labeled_function( $q', f$ );
7.       end
8.     until (For every state  $q' \in V_q$ , labeled_function( $q'$ ) does not change)
  end

```

Fig. 9. Procedure label_accept

```

procedure label.all( $q$ )
  /* ( $V_q, E_q$ ) is an SCC containing  $q$ . */
  begin
1.   repeat
2.     for all  $q' \in V_q$  do
3.       begin
4.          $f = \text{labeled\_function}(q') \vee$ 
            $\bigvee_{(q', q'') \in E_q} ( \text{BF}(\text{labeled\_function}(q'')) \wedge f_{Br(q', q'')} );$ 
5.         labeled_function( $q', f$ );
6.       end
7.     until (For every state  $q' \in V_q$ , labeled_function( $q'$ ) does not change)
  end

```

Fig. 10. Procedure label_all