NHK$^\sharp$ User's Manual

Noayuki Nagatou*
PRESYSTEMS Inc.

nagatou@presystems.xyz

Abstract

NHK$^\sharp$ is a model checking tool developed by us and published according to GPLv3 and later.

Contents

1 Installation

Compiling those files needs glib-1.2. You need to install it before doing this. Please refer the manual to install glib-1.2.

1. After installing glib-1.2,
   > tar -xvzf rccs.tar.gz

2. type the following
   > cd rccs/src

3. similarly,
   > make rccs

If you cannot compile then please edit the make file for your configuration. Perhaps, glib is installed into a different place.

2 Syntax of Model Description Language

2.1 Primitive Types

RCCS has two types, integers and strings type. Operator `+,-,/,*` are defined over these types.

2.2 Special Actions and Processes

Action `key` and `display` are special actions. `key` is used as an input action, and `display` is used as an output action. Those special actions distiguish between upper letters and lower letters.

Process `ZERO` and `STOP` are a special process that means do nothing, not terminate the whole. Those special processes do not distinguish between upper letters and lower letters.

2.3 Scope

In expression (define P (x) body), the scope of x becomes body. In expression ($\overline{a}$(x):body), the scope of x becomes body.

2.4 Lexical Binding Rule

Dynamic binding.

2.5 Syntax of the Discription Language

```
Agent_Exp     ::= ( define ID ( ID_Seq ) Agent_Exp )
               |  ( define ID ( ) Agent_Exp )
               |  ( bind ID Strings )
               |  ( globalvar IDs Strings )
               |  ( if ( B_exp ) Agent_Exp Agent_Exp )
```

```
               |  ( A_Binary_Exp )
               |  (  )
A_Binary_Exp  ::= A_Binary_Exp ++ A_Label_Exp
               |  A_Binary_Exp || A_Label_Exp
               |  A_Label_Exp
A_Label_Exp   ::= A_Label_Exp [ ID_Seq ]
               |  A_Label_Exp { Relabel_Seq }
               |  A_Unary_Exp
A_Unary_Exp   ::= ID ( Value_Seq ) : A_Unary_Exp
               |  ID : A_Unary_Exp
               |  ~ ID ( Value_Seq ) : A_Unary_Exp
               |  ~ ID : A_Unary_Exp
               |  ID ( Value_Seq )
               |  ID
               |  ( Agent_Exp )
ID_Seq        ::= ID_Seq , ID
               |  ID_Seq , ~ ID
               |  ID
               |  ~ ID
Value_Seq     ::= Value_Seq , B_Exp
               |  B_Exp
Relabel_Seq   ::= Relabel_Seq  , ID / ID
               |  Relabel_Seq  , ~ ID / ~ ID
               |  ID / ID
               |  ~ ID / ~ ID
B_Exp         ::= B_Exp | C_Exp
               |  B_Exp & C_Exp
               |  C_Exp
C_Exp         ::= C_Exp < V_Exp
               |  C_Exp <= V_Exp
               |  C_Exp > V_Exp
               |  C_Exp >= V_Exp
               |  C_Exp = V_Exp
               |  V_Exp
V_Exp         ::= V_Exp + V_Term
               |  V_Exp - V_Term
               |  V_Term
V_Term        ::= V_Term * V_Unary_Exp
               |  V_Term / V_Unary_Exp
               |  V_Term % V_Unary_Exp
               |  V_Unary_Exp
V_Unary_Exp   ::= ! V_Unary_Exp
               |  Fact
Fact          ::= Iconst | Strings | TRUE | FALSE | ID
               |  ( B_Exp )
```

2.6 RCCS Operatinal Semantics

In this section, we define the semantics of the discription language. For this purpose, we define Push-Down Automata

$$(Q, Act, \rightarrow, E, S_0)$$

where Q is a set of states, i.e. a set of processes. Act a set of actions, $\rightarrow \subseteq Q \times Act \times Q$. We write $q_1 \overset{a}{\rightarrow} q_2$ to $(q_1, a, q_2) \in \rightarrow$. E is an initial state of a model. S_0 represents a state of a stack, and the initial state of the stack is empty. Elements of the stack are pairs of a label and a value which are wrote by $a; v$, where $a \in ACT, v \in Val$. The list of pairs represents a state of the stack. The functions for the stack are defined as follows: $push(a; v, s)$ is a function that takes s and returns a list which is appended with $a; v$. $delete(a, s)$ is a function that deletes a first pair that contains a from s and return the list deleted the pair. The configuration is given by $[q, s] \in Q \times (Act \times Val)^*$.

$$\frac{}{[\sim a(v) : E, s] \overset{\sim a}{\rightarrow} [E, \mathrm{push}(a; v.s)]}\ \mathrm{Act}_1 \qquad \frac{\text{member(a,s)}}{[a(x) : E, s] \overset{a}{\rightarrow} [E, \mathrm{delete}(a, s)]}\ \mathrm{Act}_2$$

$$\frac{[E, s] \overset{\alpha}{\rightarrow} [E', s')]}{[E + +F, s] \overset{\alpha}{\rightarrow} [E', s')]}\ \mathrm{Sum}_1 \qquad \frac{[F, s] \overset{\alpha}{\rightarrow} [F', s')]}{[E + +F, s] \overset{\alpha}{\rightarrow} [F', s')]}\ \mathrm{Sum}_2$$

$$\frac{[E, s] \overset{\alpha}{\rightarrow} [E', s']}{[E||F, s] \overset{\sim\alpha}{\rightarrow} [E'||F, s']}\ \mathrm{Com}_1 \qquad \frac{[F, s] \overset{\alpha}{\rightarrow} [F', s']}{[E||F, s] \overset{\sim\alpha}{\rightarrow} [E||F', s']}\ \mathrm{Com}_2$$

$$\frac{[E||F, s] \overset{\sim\alpha}{\rightarrow} [E'||F, s'] \quad [E||F, s] \overset{\alpha}{\rightarrow} [E||F', s'']}{[E||F, s] \overset{\tau}{\rightarrow} [E'||F', s'']}\ \mathrm{Com}_3$$

$$\frac{[P, s] \overset{\alpha}{\rightarrow} [P', s'] \quad A \leftarrow P}{[A, s] \overset{\alpha}{\rightarrow} [P', s']}\ \mathrm{Con}$$

$$\frac{[E, s] \overset{\alpha}{\rightarrow} [E', s'] \quad \mathrm{eval}(B)}{[\text{if } B\ E\ F, s] \overset{\alpha}{\rightarrow} [E', s']}\ \mathrm{If}_1 \qquad \frac{[F, s] \overset{\alpha}{\rightarrow} [F', s'] \quad \neg\mathrm{eval}(B)}{[\text{if } B\ E\ F, s] \overset{\alpha}{\rightarrow} [F', s']}\ \mathrm{If}_2$$

$$\frac{[E, s] \overset{\alpha}{\rightarrow} [E', s'] \quad \alpha, \sim\alpha \in L}{[E[L], s] \overset{\alpha}{\rightarrow} [E'[L], s']}\ \mathrm{Res} \qquad \text{where L=\{a,b,}\cdots\}$$

$$\frac{[E, s] \overset{\alpha}{\rightarrow} [E', s'] \quad \alpha'/\alpha \in R}{[E\{R\}, s] \overset{\alpha'}{\rightarrow} [E'\{R\}, s']}\ \mathrm{Rel} \qquad \text{where R=\{a'/a,}\sim\text{a'/}\sim\text{a,}\cdots\}$$

$$\frac{}{[\text{define } A\ P, s] \rightarrow [\mathrm{STOP}, s]}\ \mathrm{Def}$$

$$\frac{}{[\text{bind } A\ string, s] \rightarrow [\mathrm{STOP}, s]}\ \mathrm{Bind}$$

$$\frac{}{[\mathrm{STOP}, s] \not\rightarrow}\ \mathrm{Stop}_1$$

$$\frac{[\mathrm{STOP}, s] \not\rightarrow \quad [\mathrm{STOP}, s] \not\rightarrow}{[\mathrm{STOP}||\mathrm{STOP}, s] \not\rightarrow}\ \mathrm{Stop}_2$$

$$\frac{[P, s] \overset{\alpha}{\rightarrow} [P', s']}{[\mathrm{STOP} + +P, s] \overset{\alpha}{\rightarrow} [P', s']}\ \mathrm{Stop}_3 \qquad \frac{[\mathrm{STOP}, s] \not\rightarrow \quad [\mathrm{STOP}, s] \not\rightarrow}{[\mathrm{STOP} + +\mathrm{STOP}, s] \not\rightarrow}\ \mathrm{Stop}_4$$

3 Coroutine-Like Sequencing

An important application of coroutine is discrete event simulation, where coroutine may be used to simulate parallel processes within the framework of a sequential program.

4 Syntax of Formulae

We use LTL to describe goal properties of processes. We first assume that a trace has initial states and is a finite sequence of states. We write the length of trace $\sigma = s_0\, s_1\, \cdots\, s_n$ to $|\sigma|$ in which $|\sigma|$ is $n+1$. We write the suffix of $\sigma = s_0\, s_1\, \cdots\, s_i\, \cdots\, s_n$ starting at i as $\sigma^{i..} = s_i\, \cdots\, s_n$, and the i^{th} state as σ^i.

We assume a vocabulary $x, y, z, \cdots$ of variables for data values. For each state, variables are assigned to a single value. A state formula is any well-formed first-order formula constructed over the given variables. Such state formulas are evaluated on a single state to a boolean value. If the evaluation of state formula p becomes true over s, then we write $s[p] = \texttt{tt}$ and say that s satisfies p, where `tt` and `ff` are truth values, denoting *true* and *false* respectively. Let φ and ψ be temporal formulas, a temporal formula is inductively constructed as follows:

- a state formula is a temporal formula,
- the negation of a temporal formula $\neg\varphi$ is a temporal formula,
- $\varphi \vee \psi$ and $\varphi \wedge \psi$ are temporal formulas, and
- $\Box\,\varphi$, $\Diamond\,\varphi$, $\circ\,\varphi$, and $\varphi\,\mathcal{U}\,\psi$ are temporal formulas.

We provide the formal syntax with BNF notation.

```
Start              ::= StartFormula

StartFormula       ::= StartFormula /\ PathFormula
                   |   StartFormula \/ PathFormula
                   |   StartFormula -> PathFormula
                   |   ! StartFormula
                   |   PathFormula

PathFormula        ::= <> PathFormula
                   |   [] PathFormula
                   |   PathFormula U StartFormula
                   |   X PathFormula
                   |   Proposition

Proposition        ::= Proposition &  Atom
                   |   Proposition |  Atom
                   |   Proposition -> Atom
                   |   ! Proposition
                   |   Atom

Atom               ::= Atom =  Exp
                   |   Atom <  Exp
                   |   Atom >  Exp
                   |   Atom <= Exp
                   |   Atom >= Exp
                   |   Boolean
                   |   Exp

Exp                ::= Exp + Term
                   |   Exp - Term
                   |   Term

Term               ::= Digits
                   |   Strings
                   |   Id
                   |   ( StartFormula )

Boolean            ::= tt
                   |   ff

Action             ::= Id
                   |   ~ Id
```

5 Semantics of Property Description Language

We next define two semantics of temporal formulas over a finite trace according to [EFH$^+$03]. If trace σ satisfies property φ, then we write $\sigma \models \varphi$.

5.1 Strong Semantics

Furthermore,

- if p is a state formula, then $\sigma \models p$ iff $\sigma^0[p] = \mathtt{tt}$ and $|\sigma| \neq 0$,
- $\sigma \models \neg\varphi$ iff $\sigma \not\models \varphi$,
- $\sigma \models \varphi \vee \psi$ iff $\sigma \models \varphi$ or $\sigma \models \psi$,
- $\sigma \models \varphi \wedge \psi$ iff $\sigma \models \varphi$ and $\sigma \models \psi$,
- $\sigma \models \Box\varphi$ iff for all $0 \leq i < |\sigma|$, $\sigma^{i..} \models \varphi$,
- $\sigma \models \Diamond\varphi$ iff there exists $0 \leq i < |\sigma|$ such that $\sigma^{i..} \models \varphi$,
- $\sigma \models \circ\,\varphi$ iff $\sigma' \models \varphi$ where $\sigma' = \sigma$ if $|\sigma| = 1$ and $\sigma' = \sigma^{1..}$ if $|\sigma| > 1$,
- $\sigma \models \varphi\,\mathcal{U}\,\psi$ iff there exists $0 \leq k < |\sigma|$ s.t. $\sigma \models \psi$ and for all $j < k$, $\sigma \models \varphi$.

A formula φ is satisfiable if there exists a sequence σ such that $\sigma \models \varphi$. Given set of traces T and formula φ, φ is valid over T if for all $\sigma \in T$, $\sigma \models \varphi$.

5.2 Weak Semantics

Furthermore,

- if p is a state formula, then $\sigma \models p$ iff $\sigma^0[p] = \mathtt{tt}$ or $|\sigma| = 0$,
- $\sigma \models \neg\varphi$ iff $\sigma \not\models \varphi$,
- $\sigma \models \varphi \vee \psi$ iff $\sigma \models \varphi$ or $\sigma \models \psi$,
- $\sigma \models \varphi \wedge \psi$ iff $\sigma \models \varphi$ and $\sigma \models \psi$,
- $\sigma \models \Box\varphi$ iff for all $0 \leq i < |\sigma|$, $\sigma^{i..} \models \varphi$,
- $\sigma \models \Diamond\varphi$ iff there exists $0 \leq i < |\sigma|$ such that $\sigma^{i..} \models \varphi$,
- $\sigma \models \circ\,\varphi$ iff $\sigma' \models \varphi$ where $\sigma' = \sigma$ if $|\sigma| = 1$ and $\sigma' = \sigma^{1..}$ if $|\sigma| > 1$,
- $\sigma \models \varphi\,\mathcal{U}\,\psi$ iff there exists $0 \leq k < |\sigma|$ s.t. $\sigma \models \psi$ and for all $j < k$, $\sigma \models \varphi$.

6 Relationships between Models and Formlae

In this subsection, we describe the relationship between algebraic models and LTL formulas. The modeling language enables us to pass values via input prefix $\alpha(e)$ and output prefix $\overline{\alpha}(x)$ with the same name. Execution of $\alpha(e)$ produces value v of e. Execution of $\overline{\alpha}(x)$ causes a single assignment to x. Furthermore, the execution of two actions causes atomic assignment $x := v$, that is, communication between two agents produces a new state by changing the values of the variables. This is similar to the first paragraph in Section 3.3 of [LS84, page 290].

This atomic assignment changes states, and we represent the change as $s[v/x]$, which denotes a change in the values of x in s to v. A state is a mapping from variables to values. Assuming that $\mathtt{Var_E}$ is a set of variables that appears in prefixes in agent E with range $\mathtt{V}$, $s : Var_E \to \mathtt{V}$. For example, the evaluation $s[x = y]$ of $x = y$ at s becomes $s[x] = s[y]$, and at $s[v/x]$, $s[v/x][x] = s[v/x][y]$, i.e., $v = s[y]$.

Therefore, communication between agents produces a sequence of assignments, which then produces a sequence of state changes called a trace. Let a set of traces produced by agent E be T. If for all traces $\sigma \in T$, $\sigma \models \varphi$, then we state that φ is valid over E and write $E \models \varphi$.

6.1 Transition of Automata

A Büchi automaton m contains of five components:

- A finite set of states, denoted Q.
- A finite set of input symboles, denoted Σ.
- A transition function δ that takes a state and an input symbol, and returns a next state. If q is a state, and s is an input symbol, then $\delta(q, a)$ returns state p.
- A start state q_0 is a state in Q.
- A set of accepting states Q_∞ is a subset of Q.

In this paper, an input symbol becomes a state of a model. We talk about an automaton m in *five-tuple* notation: $(Q, \Sigma, \delta, q_0, Q_\infty)$.

Now, we need to make the notion of the language that an automaton accepts. To do this, we define an extended transition function. The extended transition function constructed from δ is called $\hat{\delta}$. We define $\hat{\delta}$ by induction on the length of an input string σ, as follows:

$$\hat{\delta}(q, \sigma) = \begin{cases} \eta & \text{if } |\sigma| = 0 \\ \delta(\hat{\delta}(q, \sigma^{..n-1}), \sigma^n) & \text{if } 0 < |\sigma| < \omega. \end{cases}$$

We define the laguage $\mathcal{L}(m)$ of automaton m. Let $INF(\sigma)$ be a set of automaton states that appear infinitely often in while reading σ, then σ is accepted by m if and only if $INF(\sigma) \cup Q_\infty \neq \emptyset$. Thus,

$$\mathcal{L}(m) = \{\rho \mid \rho^0 = q^0, \rho^{|\sigma|} = \hat{\delta}(\rho, \sigma), \text{ and } INF(\sigma) \cup Q_\infty \neq \emptyset\}.$$

η depends on weak or strong semantics. In weak semantics, η is q that is regarded as an element of Q_∞. In strong semantics, η is Λ, where Λ is inconsistency.

6.2 Correspondence between Models and Formulae

We describe a correspondence between a model and a Büchi automaton of a formulae of a property which the model are required. The correspondece is expressed with Hoare triple: {P}α{P'}, where P and P' are boolean predicates, and α is an action which a model performs.

An automaton m enters an automaton state q_j if there exists a history containing program state s and m is transformed from q_i into q_j by reading s. We define *correspondence invariant* by induction [AS87].

DEFINITION 6.1 (Correspondece Basis)

$\forall i : q_j \in Q$ and $(Init_\pi \wedge T_{0j}) \Rightarrow C_j$,

where $Init_\pi$ is the initial states of model π.

DEFINITION 6.2 (Correspondece Induction)

$\forall \alpha : \forall i : \alpha \in A \cup \overline{A}$ and $q_i \in Q$ and $\{C_i\}\alpha\{\wedge_{q_j \in Q}(T_{ij} \Rightarrow C_j)\}$.

References

[AS87] Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. In *Distributd Computing*, pages 117–126. Springer-Velag, 1987.

[EFH+03] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 27–39. Springer Berlin Heidelberg, 2003.

[LS84] Leslie Lamport and Fred B. Schneider. The "Hoare Logic" of CSP, and all that. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(2):281–296, April 1984.