# Verification of Transition System Reduction via PVS

O. Takaki M. Takeyama H. Watanabe

Kyoto Sangyo University and AIST

# Verification of Transition System Reduction via PVS

O.Takaki* M.Takeyama† H.Watanabe‡

We formally verify the correctness of Transition System Reduction (TSR), an algorithm used in model-checkers for temporal logics. Formalizing TSR as a function, we formulate and prove its correctness within the proof assistant PVS. We show how to use a well-ordering on a certain set in a termination proof for the loop-based TSR algorithm. We further detail TSR's partial-correctness proof. The formal framework for these proofs is a part of our research for a rigorous verification environment for reactive systems.

## 1 Introduction

In model-checking, reactive systems are modeled as transition systems and verified against specifications written in a temporal logic. It is becoming a practical choice as a formal method, due to the availability of powerful automatic model-checkers.

Transition System Reduction (TSR) is a classic algorithm used in model checkers to minimize (Definition 2.4) a given model of a reactive system into another while preserving temporal properties. It is important as it may reduce a model whose state set is infinite or too large to be model-checked to a readily checkable small model. Kanellakis and Smolka [3] introduced TSR (there called the "naive method") generalizing the connection between minimization of finite state automata and a combinatorial partitioning problem.

Our aim is to formally verify the correctness of TSR. It is usually described in an imperative style, with somewhat complex loops that modify program states. We give a direct translation into a functional form that is suitable for formalization in PVS, formulate the correctness criteria, and give its proofs.

Section 2 recalls some basics. The algorithm TSR is introduced and formalized in Section 3. The loops in TSR are translated to recursively defined functions. This section also formulates the correctness of TSR.

The main body of this paper consists of Sections 4 and 5. Section 4 is devoted to explain how to prove the termination of TSR in PVS. We assign weights to

*Faculty of Science, Kyoto Sangyo University

†Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology (AIST)

‡Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology (AIST)

arguments of the (recursively defined) function that formalizes the main loop, and use the property that the set of weights are well-ordered. In Section 5, we show the detail of the proof of partial correctness of TSR in PVS.

Section 6 explains the actual PVS-file `min.pvs` (available at the website [7]).

These sections together are written so as to be helpful when readers examine `min.pvs`, and to give an idea on what it is like to prove a statement completely rigorously.

We also make another pvs-file for verification of TSR from the view point of formal verifications for more abstract concepts. We mention such a verification in Section 7.

This work is also motivated by our wider goal of developing a rigorous verification environment for reactive systems that combines interactive theorem proving and model checking. The expressiveness and logical strength of interactive theorem proving should make more rigorous the current practice of model checking in two respects: one is the abstraction process of an informal, high-level description of large systems into small enough models in checkers' low-level input languages; the other is verification of checkers themselves, which are, with various complex optimizations, not known to be bug-free. The framework for formalizing relevant concepts and proofs developed here should be a useful starting point for achieving the goal.

# 2 Preliminaries

TSR is used to achieve minimization of Kripke structures. Kripke structures are the standard models of reactive systems. Minimization of a Kripke structure reduces its state set while keeping it bisimilar to the original. It does so by partitioning the state set into equivalence classes of bisimilar states. We recall some basic concepts and explain these.

A *transition system* $(S, R)$ is a pair of a set $S$ and a relation $R \subseteq S \times S$.c Elements of $S$ are called its *states* and $R$ its *(total) transition relation*. For a set $AP$ of atomic propositions, a *Kripke structure* $(S, I, R, L)$ *on* $AP$ consists of a transition system $(S, R)$, a subset $I \subseteq S$ of initial states, and a labeling function $L : S \to \wp(AP)$. ($\wp(AP)$ is the powerset of $AP$.) The labeling $L$ specifies those atomic propositions that hold at each state. In this paper, we fix $AP$ and consider only those Kripke structures whose state sets are finite.

**Definition 2.1** For two Kripke structures $M_1 := (S_1, I_1, R_1, L_1)$ and $M_2 := (S_2, I_2, R_2, L_2)$, a *simulation relation from* $M_1$ *to* $M_2$ is a relation $H \subseteq S_1 \times S_2$ that satisfies the following.

(i) Whenever $H(s_1, s_2)$ and $R_1(s_1, t_1)$, there exists $t_2 \in S_2$ with $R_2(s_2, t_2)$ and $H(t_1, t_2)$.

(ii) For each $s_1 \in I_1$, there exists $s_2 \in I_2$ with $H(s_1, s_2)$.

(iii) $L_1(s_1) = L_2(s_2)$ whenever $H(s_1, s_2)$.

A *bisimulation relation $H$ between $M_1$ and $M_2$* is a simulation relation from $M_1$ to $M_2$ whose opposite $H^\circ \subseteq S_2 \times S_1$ is also a simulation relation from $M_2$ to $M_1$. $M_1$ is *bisimilar* to $M_2$ if there exists a bisimulation relation between them. When this is the case, for temporal logics such as CTL$^*$, $M_1$ satisfies a given property if and only if $M_2$ does (cf. [2]).

Fix a Kripke structure $M := (S, I, R, L)$ for the rest of this section.

A pair of states $s$ and $t$ in $S$ are *bisimilar*, written $s \simeq t$, if they are related by some bisimulation relation between $M$ and itself. The relation $\simeq$ is an equivalence relation.

Bisimilar states can be identified in the sense that the resulting Kripke structure is bisimilar to $M$. The minimization of $M$ identifies all the bisimilar states and reduces $S$ to the quotient set $S/\simeq$. We follow [3] and formulate this in terms of partitions:

**Definition 2.2** A *partition* of $S$ is a set $P \subseteq \wp(S)$ with the following conditions.

(i) $\bigcup P = S$.

(ii) Distinct $B, C \in P$ are disjoint.

(iii) $\emptyset \notin P$.

**Definition 2.3** For a partition $Q$, the *quotient* Kripke structure $M_Q$ is $(Q, I_Q, R_Q, L_Q)$ where

$$\begin{aligned} I_Q &:= \{B \in Q \mid I \cap B \neq \emptyset\} \\ R_Q &:= \{(B, C) \in Q \times Q \\ &\qquad \mid \exists\, s \in B.\ \exists\, t \in C.\ R(s,t)\} \\ L_Q &:= (B \mapsto \textstyle\bigcup_{s \in B} L(s)) \end{aligned}$$

**Definition 2.4** The *minimized*[1] structure of $M$ is the quotient Kripke structure $M_{S/\simeq}$.

We now show that $M_{S/\simeq}$ is bisimilar to $M$, using a certain property of the partition $S/\simeq$. We use some more terminology: The elements of a partition are called its *blocks*. For partitions $P$ and $Q$, we say that *$P$ is a refinement of $Q$*, or equivalently *$Q$ is coarser than $P$*, if each $B \in P$ has some $C \in Q$ with $B \subseteq C$. A pair $(B_1, B_2)$ of blocks in a partition $Q$ is *connected* if $R_Q(B_1, B_2)$. $(B_1, B_2)$ is *stable* if it is not connected, or, for each $s \in B_1$, there exists $t \in B_2$ with $R(s,t)$. We write $P_L$ for the partition induced by the labeling function $L$, that is, $P_L := S/\sim_L$ where $x \sim_L y \Leftrightarrow_{\text{def}} L(x) = L(y)$.

**Theorem 2.5** For any stable refinement $Q$ of the partition $P_L$, $M$ is bisimilar to the quotient $M_Q$.

[1] $M_{S/\simeq}$ need not be the structure with the least number of states that is bisimilar to $M$, since we do not consider the removal of unreachable states.

**Proof.** The relation $H := \{(s, B) | s \in B \in Q\}$ is a bisimulation relation between $M$ and $M_Q$. That $H$ satisfies the clause (i) of Definition 2.1 is immediate from the definition. To see the opposite $H^\circ$ satisfies (i) too, suppose that $H(s, B)$ (i.e., $s \in B$) and $R_Q(B, C)$. The stability of $(B, C)$ gives for $s$ some $t \in B$ (i.e., $H(t, B)$) with $R(s, t)$ as required. The clauses (ii) and (iii) are also immediate from the definition of $M_Q$ and that $Q$ refines $P_L$. □

**Proposition 2.6** $S/\simeq$ is a stable refinement of $P_L$.

**Proof.** That it refines $P_L$ is immediate from the clause (iii) of Definition 2.1. For the stability, suppose that $B_1$ and $B_2$ in $S/\simeq$ are connected, that is, $R(s_0, t_0)$ for some $s_0 \in B_1$ and $t_0 \in B_2$. For any $s \in B_1$, i.e., $s_0 \simeq s$, we have that $H(s_0, s)$ for some bisimulation $H$ between $M$ and itself. So, there exists $t$ such that $R(s, t)$ and $H(t_0, t)$, hence $t \in B_2$. □

So we have

**Corollary 2.7** $M_{S/\simeq}$ is bisimilar to $M$.

Together with Proposition 2.6, the following characterizes $S/\simeq$.

**Proposition 2.8** Any stable refinement $Q$ of $P_L$ is a refinement of $S/\simeq$.

**Proof.** It is enough to show that the equivalence relation $H$ such that $S/H = Q$ is a simulation relation (since $H$ is symmetric). For the clause (i) of Definition 2.1, suppose that $H(s_1, s_2)$ and $R(s_1, t_1)$. In other words, $s_2$ is in the equivalence class $[s_1] \in Q$ and $[s_1]$ is connected $[t_1]$. The stability of $Q$ gives some $t_2 \in [t_1]$ (i.e., $H(t_1, t_2)$) with $R(s_2, t_2)$. The clauses (ii) (since $H$ is reflexive) and (iii) (since $Q$ refines of $P_L$) are immediate. □

**Corollary 2.9** $S/\simeq$ is the coarsest stable refinement of $P_L$.

Thus the problem of minimizing $M$ is reduced to that of constructing the coarsest stable refinement of $P_L$. $TSR$ is an algorithm to compute this refinement.

# 3 TSR and its formalization

This section introduces the algorithm TSR and shows how we formalize it.

## 3.1 Informal definition of TSR and the correctness criteria

TSR takes as an input a transition system $(S, R)$ and a partition $P$ on $S$, and returns the coarsest stable refinement of $P$. We fix an arbitrary $(S, R)$ for the rest of the paper and suppress its mention as an input.

A partition is represented by the list of its blocks. We use the following list operations: for list $L := (x_1, x_2, \ldots, x_n)$, $car(L) := x_1$, $cdr(L) := (x_2, \ldots, x_n)$, $enqueue(L, x) := (x_1, x_2, \ldots, x_n, x)$, $append(L, (y_1, \ldots, y_n)) := (x_1, \ldots, x_n, y_1, \ldots, y_n)$,

$rotate(L) := (x_2, \ldots, x_n, x_1)$. The length of $L$ is written $lg(L)$. For a block $B$, we write $Pre(B)$ for the set $\{x \in S \mid \exists y \in B.\, R(x, y)\}$ of predecessor states.

At an abstract level, TSR proceeds as follows: It maintains the 'current' partition (the list $L$ in the pseudo-code) initialized to the input $P$. While there is an unstable pair $(C, B)$ of blocks in $L$, it refines $L$ by splitting $C$ into two blocks $C'$ and $C''$ so that $(C', B)$ is connected and stable and $(C'', B)$ is not connected (hence stable). It terminates when there is no unstable pair in $L$ and outputs $L$.

The pseudo-code below is a particular implementation (a modified version of the one given in [4]).

```
Algorithm TSR
  Input: (P: partition on S)
  Output: (L: list of subsets of S)
begin
   L := list of the blocks in P;
   n := lg(L);
   while (n > 0) do
      B := car(L);
      l := lg(L);
      for l times do
         L := append(cdr(L), split(car(L), B))
      end
      if (l ≠ lg(L)) then n := lg(L);
                     else n := n − 1;
                          L := rotate(L)
      endif
   end
end
```

Splitting of a block $C$ by $B$ is done by the function *split*:

$$
\begin{aligned}
split(C, B) := \ & \text{IF } (C' \neq \emptyset \text{ and } C' \neq C) \\
& \quad \text{THEN } (C',\ C - C') \\
& \quad \text{ELSE } (C) \\
& \text{where } C' = C \cap Pre(B).
\end{aligned}
$$

When $(C, B)$ is already stable, it returns the singleton list $(C)$ so that it becomes a no-operation in this case.

Rather than searching for an unstable pair, we apply *split* to all the pairs $(C, B)$ anyway; the outer while-loop changes $B$ and the inner for-loop changes $C$. TSR terminates when it detects that no actual splitting occurred while all the pairs are tried.

The variable $n$ maintains the number of remaining blocks that should be tried as $B$ before we can say that all the pairs $(C, B)$ are tried. Initially, $n$ is the number of all blocks in $L$. TSR terminates when $n$ becomes 0. When $n > 0$, TSR enters the body of the outer while-loop. At this point, the blocks to be

tried as $B$ are the $n$ foremost (leftmost) blocks in $L$. It chooses $car(L)$ as $B$. Let $l$ be the number of blocks currently in $L$. The inner for-loop splits those $l$ blocks by $B$ and stabilizes $L$ with respect to $B$. Its one step takes out a block from the front of $L$ and put its splitting by $B$ at the back. If the block taken out from the front is already stable with respect to $B$, this is the same as rotating $L$ by one element. After this is done for $l$ times, it is either the case that some actual splitting occurred and $L$ now contains more blocks ($l \neq lg(L)$), or that no splitting occured and $L$ is rotated back all the way to the same list as before. In the former case, we reset $n$ to the number of blocks in the current $L$, that is to say we start over with the current $L$ as the input partition. In the latter case, we decrease $n$ and rotate $L$ so that new blocks to be tried as $B$ come at the foremost part of $L$, and continue the while-loop.

The correctness of TSR is split into two parts:

**Termination of TSR**: TSR terminates for any input partition $P$; that is, the while-loop terminates after finitely many steps.

**Partial correctness of TSR**: For the output list $\pi$ of TSR on input $P$,

(i) $\pi$ represents a partition of $S$;

(ii) $\pi$ is a refinement of $P$;

(iii) $\pi$ is stable; and

(iv) $pi$ is the coarsest partition satisfying the properties (ii) and (iii).

## 3.2 Formalization of TSR and the correctness

We now formalize TSR and its correctness essentially in the language of PVS, though we use sugared notation for readability. We reformulate TSR as a function, as PVS's language is a functional one. We aim to formalize the imperative style as directly as possible. As in the previous section, we keep $(S, R)$ fixed and implicit as a part of an input.

We fix some notation: For sets $X$ and $Y$, $[X, Y]$ denotes their cartesian product, $[X \to Y]$ the set of functions from $X$ to $Y$, and LIST($X$) the set of lists whose elements are in $X$. $\mathbb{N}$ is the set of natural numbers.

The $n$-ary product $[\cdots[X_1, X_2], X_3], \cdots, X_n]$ is abbreviated to $[X_1, \ldots, X_n]$, and the curried $(n-1)$-ary function set $[X_1 \to [X_2 \to \cdots [X_{n-1} \to X_n] \cdots]]$ to $[X_1 \to X_2 \to \cdots X_{n-1} \to X_n]$.

For an element $s$ of $[X_1, \ldots, X_n]$, the $i^{\text{th}}$-component of $s$ is written $pr_i(s)$. We write $x \,\varepsilon\, L$ to mean that a list $L$ contains an element $x$. Finally, we write $\mathbb{B}$ for $\wp(S)$ ('blocks') and $\mathbb{L}$ for LIST($\mathbb{B}$).

The formalization of 'a list representing a partition' is as follows.

**Definition 3.1** A *partition list* of a set $S$ is an $L \in \mathbb{L}$ satisfying:

(i) for each $x \in S$, there exists $B \,\varepsilon\, L$ with $x \in B$;

(ii) for each $n, m < lg(L)$, $n \neq m$ implies that $n$-th and $m$-th elements of $L$ are disjoint; and

(iii) $\emptyset \notin L$.

The subset of $\mathbb{L}$ given by partition lists is denoted by $\mathbb{PL}$. The function that formalizes TSR is defined for $\mathbb{L}$, however.

A general principle for a functional reformulation is to regard an imperative statement as a function on program states. Here we mean by a program state an assignment of values to all program variables. If the execution of a statement in a program state $\sigma$ results in a program state $\sigma'$, the corresponding function sends $\sigma$ to $\sigma'$. We follow this principle, but our functions take and return only (tuples of) values that are assigned to relevant program variables. The way these functions are composed reflects which value is intended for which program variable.

We formalize a loop in the imperative style as an iterated application of a 'step function' $f$ that corresponds to the body (one step) of the loop. We use the function $iter(f, x, n)$, the $n$-th iterated application of $f$ on $x$ (i.e., $f^n(x)$).

$$
\begin{array}{l}
iter : [[X \to X], X, \mathbb{N}] \to X \\
iter(f, x, n) := \text{IF } (n = 0) \text{ THEN } x \\
\qquad\qquad\quad \text{ELSE } f(iter(f, x, n-1))
\end{array}
$$

The detail of the reformulation of the inner for-loop and the outer while-loop is as follows.

The body of the for-loop modifies the list assigned to the program variable '$L$'. The corresponding step function $split1(B)$ of type $\mathbb{L} \to \mathbb{L}$ formalizes the modification. Here $B$ is the block assigned to the program variable '$B$' referred in the body; it is not changed in the body, so treated as a parameter. The for-loop as a whole is $stabilize(B)$.

$$
\begin{array}{l}
split1 : \mathbb{B} \to [\mathbb{L} \to \mathbb{L}] \\
split1(B)(L) := \text{ IF } (L = \text{ null}) \text{ THEN } L \\
\qquad\qquad\quad \text{ELSE } append(cdr(L), \\
\qquad\qquad\qquad\qquad\quad split(car(L), B))
\end{array}
$$

$$
\begin{array}{l}
stabilize : \mathbb{B} \to [\mathbb{L} \to \mathbb{L}] \\
stabilize(B)(L) := iter(split1(B), L, lg(L))
\end{array}
$$

The value of $split1(B)$(null) can be an arbitrary dummy value; it is still defined since total functions are easier to deal with in PVS.

The body of the outer while-loop modifies the list and number assigned to the program variables '$L$' and '$n$'. The step function *step* takes and returns values of type $[\mathbb{L}, \mathbb{N}]$ to represent the modification.

$$
\begin{array}{l}
step : [\mathbb{L}, \mathbb{N}] \to [\mathbb{L}, \mathbb{N}] \\
step(L, n) := \\
\quad \text{IF } (L = \text{ null OR } n = 0) \text{ THEN } (L, 0)
\end{array}
$$

ELSE LET $V := stabilize(car(L))(L)$
IN IF $(L \neq V)^2$ THEN $(V,\ lg(V))$
ELSE $(rotate(L),\ n-1)$

The function *funTSR* formalizes the outer while-loop, that is to say the whole of TSR.

$funTSR : \mathbb{L} \to [\mathbb{L}, bool]$
$funTSR(L) :=$
LET $steps : \mathbb{N} \to [\mathbb{L}, \mathbb{N}]$
$steps(k) := iter(step,\ (L, lg(L)),\ k);$
$L_k := pr_1(steps(k))$
$n_k := pr_2(steps(k))$
$K := \{k \in \mathbb{N} \mid n_k = 0\}$
IN IF $(K \neq \emptyset)$
THEN $(L_{min(K)},\ true)$
ELSE $(L,\ false)$

We do not know a priori whether the loop terminates for a given initial list $L$. So we let *funTSR* return $(L', true)$ if the loop does terminate with the result list $L'$, and $(L, false)$ if it does not ($L$ in this case is a dummy value). The list $L_k$ and the number $n_k$ are those assigned to the program variables '$L$' and '$n$' after $k$-th iteration of the while-loop. To see whether the loop terminates, we check if there are $k$ such that $n_k = 0$. If there are, then the minimum among such, $min(K)$ in the code, is the number of iterations for the loop. So the result list is $L_{min(K)}$.

We now formalize the correctness of of TSR given in Section 3.1.

**(TM)** (TSR terminates.)
$\forall L \in \mathbb{L}.\ pr_2(funTSR(L)) = true.$

**(PC)** (partial correctness.)
For each $L \in \mathbb{L}$, the first component $\pi$ of $funTSR(L)$ satisfies the following properties:

**(PC1)** ($\pi$ represents a partition.)

$$L \in \mathbb{PL} \ \Rightarrow\ \pi \in \mathbb{PL}$$

**(PC2)** ($\pi$ is a refinement of $L$.)

$$\begin{aligned}&\mathbf{RF}(\pi, L) \text{ where}\\ &\mathbf{RF}(Q, L) := \forall B \in \mathbb{B}.\, B\ \varepsilon\ Q \Rightarrow\\ &\qquad\qquad \exists C \in \mathbb{B}.\, C\ \varepsilon\ L\ \wedge\ B \subseteq C\end{aligned}$$

[2] We modified the condition ($lg(L) \neq lg(V)$) in the imperative version in order to simplify proofs somewhat, but dealing with the original condition does not present an essential difficulty.

**(PC3)** ($\pi$ is stable.)

$$\begin{array}{l}\mathbf{ST}(\pi) \text{ where}\\ \mathbf{ST}(Q) := \forall B, C \in \mathbb{B}.\, B\ \varepsilon\ Q\ \wedge\ C\ \varepsilon\ Q\ \Rightarrow\\ \qquad\qquad (C \cap \mathit{Pre}(B) = \emptyset \vee\\ \qquad\qquad\ C \cap \mathit{Pre}(B) = C)\end{array}$$

**(PC4)** ($\pi$ is the coarsest such.)

$$\begin{array}{l}\forall Q \in \mathbb{L}.\\ \mathbf{RF}(Q, L) \wedge \mathbf{ST}(Q)\ \Rightarrow\ \mathbf{RF}(Q, \pi).\end{array}$$

The partial correctness of TSR is in fact (PC1–4) under the assumption that $pr_2(\mathit{funTSR}(L)) = \mathit{true}$. However, our proofs use (TM) to prove (PC1–4) without the assumption.

# 4 Verification of termination

The termination of TSR, (TM), is an immediate consequence of Theorem 4.1 bellow. This section shows how to formally prove this using an well-order on $[\mathbb{L}, \mathbb{N}]$. We use the following abbreviations:

$$\begin{array}{l} f^n(x) := \mathit{iter}(f, x, n);\\ \mathit{steps}_{\mathbb{L}}(n, L, k) := pr_1(\mathit{step}^n(L, k));\\ \mathit{steps}_{\mathbb{N}}(n, L, k) := pr_2(\mathit{step}^n(L, k)).\end{array}$$

**Theorem 4.1** For any list $L \in \mathbb{L}$, there exists $n \in \mathbb{N}$ such that $\mathit{steps}_{\mathbb{N}}(n, L, lg(L)) = 0$.

## 4.1 Termination and well-order

We prove Theorem 4.1 by exhibiting a well-founded relation $\sqsubset$ on $[\mathbb{L}, \mathbb{N}]$ such that

$$k \neq 0\ \Rightarrow\ \mathit{step}(L, k) \sqsubset (L, k). \tag{1}$$

The well-foundedness of $\sqsubset$ means that there is no infinite sequence of the form

$$(L_0, k_0) \sqsupset (L_1, k_1) \sqsupset \cdots .$$

The proof of Theorem 4.1 is by contradiction: If the theorem fails for some $L$, then we have the infinite sequence

$$(L, lg(L)) \sqsupset \mathit{step}(L, lg(L)) \sqsupset \mathit{step}^2(L, lg(L)) \sqsupset \cdots ,$$

which contradics the well-foundedness of $\sqsubset$.

We define $\sqsubset$ on $[\mathbb{L}, \mathbb{N}]$ using an well-order on another set, which simplifies our proofs. In more detail, we proceed as follows:

(i) Constructing a well ordered set $(Od, \prec)$.

(ii) Proving the well-foundedness of $Od$.

(iii) Constructing a weight function $W : [\mathbb{L}, \mathbb{N}] \to Od$.

(iv) Proving the property (1) for the relation

$$(L, k) \sqsubset (L', k') \quad := \quad W(L, k) \prec W(L', k').$$

## 4.2 Well-order *Od*

We define a well ordered set $(Od, \prec)$, as follows.

**Definition 4.2** (i) $Od$ is the set of strings of the form 0 or $\omega^{i_0} + \omega^{i_1} + \cdots + \omega^{i_n} + 0$, where $i_0, \cdots, i_n$ are integers with $i_0 \geqq \cdots \geqq i_n$ and $n \geqq 0$.

For each $\alpha \in Od$ with $\alpha \neq 0$, there is a unique integer $i$ and $\beta \in Od$ with $\alpha = \omega^i + \beta$. The $i$ is denoted by $exp(\alpha)$ and the $\beta$ by $rest(\alpha)$.
(ii) $\prec$ is the order on $Od$ defined by

1. $0 \prec \beta$ for $\beta \neq 0$;
2. $\omega^i + \beta \prec \omega^j + \gamma$ if $i < j$, or if $i = j$ and $\beta \prec \gamma$.

For each $\alpha, \beta \in Od$, $\alpha \succ \beta$ denotes $\beta \prec \alpha$.

The order $\prec$ is well known as the *lexicographic order* on strings of integers. In the actual formalization, we do not directly define $Od$ in PVS but employ an ordinal notation system (a well-order) described in a library file attached to the standard PVS package. This saves us the effort to prove the well-foundedness of $Od$ in PVS.

## 4.3 Assignment of $[\mathbb{L}, \mathbb{N}]$ to *Od*

We here define a function $W : [\mathbb{L}, \mathbb{N}] \to Od$ of (iii) in Section 4.1.

For each $n, m \in \mathbb{N}$, we define $n \cdot \omega^m \in Od$ by

$$\begin{aligned} n \cdot \omega^m = \text{IF } (n = 0) \text{ THEN } 0 \\ \text{ELSE } \omega^m + (n-1) \cdot \omega^m. \end{aligned}$$

We also define $\oplus : [Od, Od] \to Od$ by

$$\begin{aligned} \alpha \oplus \beta = \text{IF } (\ \alpha = 0 \text{ or } exp(\alpha) < exp(\beta)\ ) \\ \text{THEN } \beta \\ \text{ELSE } \ \omega^{exp(\alpha)} + (rest(\alpha) \oplus \beta). \end{aligned}$$

For each $L \in \mathbb{L}$, a *max element* of $L$ is an element $B$ of $L$ such that the number of elements of $B$ is equal to or larger than that of any element of $L$.

**Definition 4.3** We define a function $W(L, n)$ by recursion on $L$:

$W : [\mathbb{L}, \mathbb{N}] \to Od$
$W(L, n) =$ IF $L$ is null THEN $n \cdot \omega^0$
ELSE
LET $m_0 =$ the number of elements of a max element of $L$
$n_0 =$ the number of all max elements of $L$
$L_0 =$ the list obtained from $L$ by removing all max elements of $L$
IN $n_0 \cdot \omega^{m_0+1} \oplus W(L_0, n)$.

**Example 4.4** Let $S = \{a, b, c, d, e, f, g, h\}$ and $L = (\{a, b\}, \{c, d, e\}, \{f\}, \{g, h\})$. Then, $W(L, 3) = \omega^4 + \omega^3 + \omega^3 + \omega^2 + \omega^0 + \omega^0 + \omega^0 + 0$.

In order to show (iv) in Section 4.1, we introduce some basic properties of $W$, as follows.

**Lemma 4.5** For each $L \in \mathbb{L}$ and $n \in \mathbb{N}$, we have the following properties:

1. if $n > 0$ then $W(L, n) \succ W(L, n-1)$;
2. if $car(L)$ is non-empty and divided into two non-empty subsets $B_1$ and $B_2$ of $S$, then $W(L, n) \succ W(enq(enq(cdr(L), B_1), B_2), m)$; and
3. $W(L, n) = W(rotate(L), n)$.

We here show the property (iv) in Section 4.1.

**Lemma 4.6** For each $L \in \mathbb{L}$ and $n \in \mathbb{N}$, if $n > 0$, then $W(step(L, n)) \prec W(L, n)$.

**Proof.** By Lemma 4.5, for each $L \in \mathbb{L}$, $B \subset S$ and $n, m \in \mathbb{N}$, we have the following properties:

(i) $W(split1(B)^m(L),\ n) \preceq W(L, n)$;

(ii) if $W(split1(B)^m(L),\ n) = W(L, n)$ then $split1(B)^m(L) = rotate^m(L)$;

(iii) if $W(split1(B)^m(L), n) \prec W(L, n)$, then, for each $k \in \mathbb{N}$, $W(split1(B)^m(L), k) \prec W(L, n)$.

Let $L$ be a list and $n$ an integer $> 0$, and set $V := stabilize(car(L))(L)$. We show the result in the following two cases.
(Case 1) Assume that no element $C$ of $L$ is divided into two non-empty blocks $C \cap Pre(car(L))$ and $C - C \cap Pre(car(L))$. Then, by the definitions of *stabilize* and *split1*, $V = rotate^{lg(L)}(L)$. Since $rotate^{lg(L)}(L) = L$, $pr_1(step(L, n)) = rotate(L)$ and $pr_2(step(L, n)) = n - 1$. Thus, by Lemmas 4.5.1 and 4.5.3, we have $W(step(L, n)) \prec W(L, n)$.

(Case 2) Assume that some element of $L$ is divided into the two blocks. Then, by the definitions of *stabilize* and *split1*, $V \neq rotate^{lg(L)}(L) = L$. Then, by (i) and (ii), $W(V,n) \prec W(L,n)$. On the other hand, $step(L,n) = V$ since $L \neq V$. Therefore, by (iii), $W(step(L,n)) \prec W(L,n)$. □

By using Lemma 4.6, we can formally prove termination of *funTSR*, according to the way which we explained in Section 4.1.

### 4.4 Library for ordinal notation systems

In our actual work via PVS, we employ a formalized ordinal notation system (= some kind of a well ordered set) named `ordinal` in the library `prelude.pvs`, instead of defining *Od* directly. It should be significant to develop a library for an ordinal notation system which has a sufficiently large scale to assure well-foundedness of orders on various mathematical structures, and to show the well-foundedness of such a system rigorously.

One of the reasons why (the library of) such a notation system $\mathcal{N}$ is useful is that one can formally show well-foundedness of each order $\mathcal{O}$ by constructing an order-preserving function from $\mathcal{O}$ to $\mathcal{N}$ as well as one can "create" a well-order on each structure $\mathcal{S}$ by constructing a function from $\mathcal{S}$ to $\mathcal{N}$ just like we do for the verification of TSR.

One might consider that *Od* has the unnecessarily large scale. Indeed, considering that $S$ is finite, we can establish a mapping: $[\mathbb{L}, \mathbb{N}] \to \mathbb{N}$ satisfying the properties mentioned in Section 4.1. However, such a function will force us to deal with much more complicated work when we try to prove termination of TSR according to the way we explained in Section 4.1. We hope that in a lot of situations we can make verifications of termination much easier by considering well orders with sufficiently large scales.

## 5 Verification of partial correctness of TSR

In this section, we explain the verification of the partial correctness of TSR, which is formalized in "theory-parts" named "`correctness_of_TSR_1`"∼ "`correctness_of_TSR_4`" in our pvs-file named "`tsr.pvs`" in [7]. We explain the proof in a manner that is convenient to read our pvs-file.

The lines of the proofs of (PC1) and (PC2) in Section 3.2 are similar to that of (PC4). So, we show (PC3) in Section 5.1 and show (PC4) in Section 5.2.

### 5.1 Verification of stability of return values of TSR

In this section, we show (PC3):

$$\text{For each } L \in \mathbb{L},\ \mathbf{ST}(pr_1(funTSR(L))).$$

That is, for each $L \in \mathbb{L}$ and $B, C\ \varepsilon\ pr_1(funTSR(L))$,

$$B \cap Pre(C) = \emptyset \ \text{ or } \ B \cap Pre(C) = B.$$

We first show an outline of the proof of (PC3), as follows. If $L$ is not null and $pr_2(funTSR(L))$ is true, we have the number $n_0$ satisfying that $steps_{\mathbb{N}}(n_0, L, lg(L)) = 0$ and $\forall m < n_0\ steps_{\mathbb{N}}(m, L, lg(L)) > 0$. So, on each step between the $(n_0 - lg(steps_{\mathbb{L}}(n_0, L, lg(L))))^{\text{th}}$ step and the $n_0^{\text{th}}$ step, *step* only rotate the given list. This implies that, for $Q := steps_{\mathbb{L}}(n_0 - lg(steps_{\mathbb{L}}(n_0, L, lg(L))), L, lg(L))$ and for each $B\ \varepsilon\ Q$, $B$ does not split any element of $Q$. So, $Q$ is stable, and hence, we have (PC3) since $Q$ is turn to be $pr_1(funTSR(L))$.

In the rest of this section, we show (PC3) in more detail, according to the theory part named "`correctness_of_TSR_3`" described in our pvs-file.

We first describe the stability **ST** by using *rotate*, as follows.

**Lemma 5.1** For each $L \in \mathbb{L}$, $\mathbf{ST}(L)$ follows from the property:

$$\begin{aligned}\forall n <& lg(L)\ \forall B\ \varepsilon\ L\\ &(\ B \cap Pre(car(rotate^n(L))) = \emptyset\\ &\qquad \text{or}\ \ B \cap Pre(car(rotate^n(L))) = B\ ).\end{aligned}$$

Next we want to examine the return value $step^{\,(n+i-lg(steps_{\mathbb{L}}(n,L,lg(L))))}(L, lg(L))$ for each $i \in \mathbb{N}$. However, in order to consider such return values, we have to check that

$$n - lg(steps_{\mathbb{L}}(n, L, lg(L))) \geqq 0.$$

**Lemma 5.2** For each $L \in \mathbb{L}$ and $n \in \mathbb{N}$, if $steps_{\mathbb{N}}(n, L, lg(L)) = 0$, then

$$lg(steps_{\mathbb{L}}(n, L, lg(L))) \leqq n.$$

**Proof.** For each $L \in \mathbb{L}$ and $k \in \mathbb{N}$, if $lg(pr_1(step(L, k))) > lg(L)$, then

$$pr_2(step(L, k)) = lg(pr_1(step(L, lg(L)))).$$

Therefore, one can show that, for each $n \in \mathbb{N}$, $n < lg(steps_{\mathbb{L}}(n, L, lg(L)))$ implies

$$steps_{\mathbb{N}}(n, L, lg(L)) \geqq lg(steps_{\mathbb{L}}(n, L, lg(L))) - n \tag{2}$$

by induction on $n$.

Suppose that $steps_{\mathbb{N}}(n, L, lg(L)) = 0$ but that $lg(steps_{\mathbb{L}}(n, L, lg(L))) > n$. Then, by (2), we have $steps_{\mathbb{N}}(n, L, lg(L)) = 0 \geqq lg(steps_{\mathbb{L}}(n, L, lg(L))) - n$, and hence, $lg(steps_{\mathbb{L}}(n, L, lg(L))) \leqq n$. This is a contradiction. □

Lemma 5.2 assures that we can consider

$$step^{\,(n+i-lg(steps_{\mathbb{L}}(n,L,lg(L))))}(L, lg(L))$$

for each $i \in \mathbb{N}$ whenever $steps_{\mathbb{N}}(n, L, lg(L)) = 0$.

We next show that, on each step between the $(n_0 - lg(steps_{\mathbb{L}}(n_0, L, lg(L))))^{\text{th}}$ step and the $n_0^{\text{th}}$ step, *step* only rotate the given list, where $n_0$ is the number in the outline of the proof of (PC3) above.

**Lemma 5.3** For each $L \in \mathbb{L}$ and $n \in \mathbb{N}$, if $steps_{\mathbb{N}}(n, L, lg(L)) = 0$ and $\forall m < n$ $(steps_{\mathbb{N}}(m, L, lg(L)) > 0)$, then, for each $i \leqq l_0 := lg(steps_{\mathbb{L}}(n, L, lg(L)))$,

$$steps_{\mathbb{L}}(n + i - l_0, L, lg(L)) = \\ rotate^i(steps_{\mathbb{L}}(n - l_0, L, lg(L))).$$

**Proof.** Let $n_0$ be the least number satisfying that $steps_{\mathbb{N}}(n_0, L, lg(L)) = 0$, set $l_0 := lg(steps_{\mathbb{L}}(n_0, L, lg(L)))$ and set $n_1 := n_0 - l_0$. Then, by Lemma 5.2, $n_1 \geqq 0$. Set $L_i := steps_{\mathbb{L}}(n_1 + i, L, lg(L))$ and $k_i := steps_{\mathbb{N}}(n_1 + i, L, lg(L))$ for each $i \leqq l_0$. Then, for each $i < l_0$, $(k_{i+1} = k_i - 1$ & $L_{i+1} = rotate(L_i))$ or $k_{i+1} = lg(L_{i+1})$. Suppose that there exists $i$ satisfying that

$$i < l_0 \ \ \& \ \ k_{i+1} = lg(L_{i+1}). \tag{3}$$

Let $I$ be the biggest number satisfying (3). For each $j$ with $I < j < l_0$, $k_{j+1} = k_j - 1$ and $L_{j+1} = rotate(L_j)$. So, $k_{I+1} = lg(L_{I+1}) = l_0$ and $steps_{\mathbb{N}}(n_0, L, lg(L)) = k_{l_0} = k_{I+1} - (l_0 - I - 1) = I + 1 > 0$. This is a contradiction. Therefore, for each $i < l_0$, $k_{i+1} = k_i - 1$ and $L_{i+1} = rotate(L_i)$. So, we have the result. □

Next we again consider the relationship between the functional *split1* and *step*. For each list $L$ and $B \subset S$, it holds that

- $lg(split1(B)(L)) \geqq lg(L)$; and
- $lg(split1(B)(L)) = lg(L)$ implies $car(L) \cap Pre(B) = \emptyset$ or $B$.

So, for each $i \leqq lg(L)$, $split1(car(L))^i(L) = rotate^i(L)$ means that, for each $j < i$, $B_j \cap Pre(car(L)) = \emptyset$ or $B_j$, where $B_j$ denotes the $j^{\text{th}}$ elements of $L$. So, we have the following lemma.

**Lemma 5.4** For each $L \in \mathbb{L}$ and $k \in \mathbb{N}$, if $L$ is not null, $k > 0$ and if $pr_1(step(L, k)) = rotate(L)$, then $split1(car(L))^{lg(L)}(L) = L$, and hence, $B \cap Pre(car(L)) = \emptyset$ or $B$ for each $B \, \varepsilon \, L$.

The next lemma is described so that one can utilize Lemma 5.4 as directly as possible to prove it.

**Lemma 5.5** Let $L \in \mathbb{L}$ and $n, k \in \mathbb{N}$. If $L$ is not null, $steps_{\mathbb{N}}(i, L, k) > 0$ for each $i < n$, and if $steps_{\mathbb{L}}(i, L, k) = rotate^i(L)$ for each $i \leqq n$, then, for each $j < n$ and $B \, \varepsilon \, L$, $B \cap Pre(car(rotate^j(L))) = \emptyset$ or $B \cap Pre(car(rotate^j(L))) = B$.

**Proof.** By induction on $n$ and Lemma 5.4. □

Now, by using Lemmas 5.1∼5.5, we prove (PC3), as follows.

The case where $L$ is null is trivial.

Suppose that $L \neq$ null and that $pr_2(funTSR(L))$ is *true*. Then, there exists $n \in \mathbb{N}$ with $step^n(L, lg(L)) = 0$. So, we can take the number $n_0$ satisfying the following properties.

$$steps_{\mathbb{N}}(n_0, L, lg(L)) = 0; \tag{4}$$

$$\forall m < n_0 \ (\ steps_{\mathbb{N}}(m, L, lg(L)) > 0\ ). \tag{5}$$

Set $L_0 := steps_{\mathbb{L}}(n_0, L, lg(L))$ and $l_0 := lg(L_0)$. Then, by Lemma 5.3, (4) and (5) above,

$$\begin{aligned}\forall i \leqq l_0 \ (\ & steps_{\mathbb{L}}(n_0 + i - l_0, L, lg(L)) \\ & = rotate^i(steps_{\mathbb{L}}(n_0 - l_0, L, lg(L)))\ ).\end{aligned} \tag{6}$$

Set $L_1 := steps_{\mathbb{L}}(n_0 - l_0, L, lg(L)))$ and $k_1 := steps_{\mathbb{N}}(n_0 - l_0, L, lg(L)))$. Then, (5) implies (7) below as well as (6) implies (8) below.

$$\forall i < l_0 \ (\ steps_{\mathbb{N}}(i, L_1, k_1) > 0\ ) \tag{7}$$

$$\forall i \leqq l_0 \ (\ steps_{\mathbb{L}}(i, L_1, k_1) = rotate^i(L_1)\ ) \tag{8}$$

On the other hand, for each $L'$, $k$ and $k'$, if $steps_{\mathbb{L}}(k, L', k')$ is null, then $L'$ is null. So, $L_1$ is not null since $L$ is not null. Therefore, by Lemma 5.5, (7) and (8),

$$\begin{aligned}&\forall j < l_0\ \forall B\ \varepsilon\ L_1 \\ &\quad (\ B \cap Pre(car(rotate^j(L_1))) = \emptyset \\ &\qquad \text{or } B \cap Pre(car(rotate^j(L_1))) = B\ ).\end{aligned} \tag{9}$$

On the other hand, (6) implies that $L_0 = rotate^{l_0}(L_1)$, which implies $l_0 = lg(L_0) = lg(L_1)$, and hence,

$$L_0 = L_1. \tag{10}$$

Thus, by (9) and Lemma 5.1, we have $\mathbf{ST}(L_0)$. By the definition of $funTSR$, $L_0 = pr_1(funTSR(L))$. So, we have (PC3).

## 5.2 Partial correctness of TSR besides stability

(PC1), (PC2) and (PC4) share a similar way to prove, that is, they are proved according to "refining path", as follows.

Let $L \in \mathbb{L}$. Then, by Theorem 4.1, there exists the least number $n_0$ satisfying that $steps_{\mathbb{N}}(n_0, L, lg(L)) = 0$ and $steps_{\mathbb{L}}(n_0, L, lg(L)) = pr_1(funTSR(L))$. So, we have the following path (with length $n_0 + 1$):

$$\begin{aligned}(L, lg(L)) \ \longrightarrow\ & step(L, lg(L)) \ \longrightarrow\ \cdots \\ & \cdots \ \longrightarrow\ step^{\,n_0}(L, lg(L)).\end{aligned}$$

This path is denoted by $path(L)$ and its length by $lp(L)$ [3].

[3] Actually, we do not need Theorem 4.1 to show (PC1), (PC2) and (PC4). Indeed, for each $L$ having no finite refinement path, one can easily check (PC1), (PC2) and (PC4), since $pr_1(funTSR(L)) = L$ (cf. (PC1)∼(PC4) in Section 3.2).

Let $\mathbf{P}$ denote a predicate on $[[\mathbb{L} \to \mathbb{L}],\ \mathbb{L}]$, and set

$$\begin{aligned} id &:= \text{the identity function on } \mathbb{L};\\ \mathbf{sp}\langle i\rangle &:= \lambda L \in \mathbb{L}.\ \mathit{iter}(\mathit{split1}(\mathit{car}(L)), L, i);\\ \mathbf{st}\langle k\rangle &:= \lambda L \in \mathbb{L}.\ \mathit{pr}_1(\mathit{step}(L, k));\\ \mathbf{ist}\langle n\rangle &:= \lambda L \in \mathbb{L}.\ \mathit{steps}_{\mathbb{L}}(n, L, \mathit{lg}(L));\ \text{and}\\ \mathbf{f}_{TSR} &:= \lambda L \in \mathbb{L}.\ \mathit{pr}_1(\mathit{funTSR}(L)). \end{aligned}$$

If $\mathbf{P}$ satisfies the following:

$$\forall L \in \mathbb{L}\ \forall k \in \mathbb{N}\ (\mathbf{P}(id, L) \Rightarrow \mathbf{P}(\mathbf{st}\langle k\rangle, L)), \tag{11}$$

then we have

$$\mathbf{P}(id, L) \Rightarrow \mathbf{P}(\mathbf{f}_{TSR}, L) \tag{12}$$

for each $L$. We can prove (12) by showing that, for each $n < \mathit{lp}(L)$, $\mathbf{P}(id, L) \Rightarrow \mathbf{P}(\mathbf{ist}\langle n\rangle, L)$, by induction on $n$. Moreover, *step* satisfies that, for each $L \in \mathbb{L}$ and $k \in \mathbb{N}$, $\mathbf{st}\langle k\rangle = L$, $\mathbf{st}\langle k\rangle = \mathit{rotate}(L)$ or $\mathbf{st}\langle k\rangle = \mathit{split1}(\mathit{car}(L))^{\mathit{lg}(L)}(L)$. Therefore, in order to show (11), it suffices to show

$$\forall\ L \in \mathbb{L}\ (\ \mathbf{P}(id, L)\ \Rightarrow\ \mathbf{P}(\mathit{rotate}, L)\ ); \tag{13}$$

$$\forall L \in \mathbb{L}\ \forall i \leqq \mathit{lg}(L)\ (\mathbf{P}(id, L) \Rightarrow \mathbf{P}(\mathbf{sp}\langle i\rangle, L)). \tag{14}$$

Thus, the main bodies to prove (PC1), (PC2) and (PC4) are to prove (13) and (14) in the case where $\mathbf{P}$ is replaced by a suitable predicate to prove (PC1), (PC2) and (PC4) respectively.

For example, we show (PC4) by using the way above, as follows.

$(PC4)$ means that, for each $L, Q \in \mathbb{PL}$,

$$\mathbf{RF}(Q, L)\ \&\ \mathbf{ST}(Q)\ \Rightarrow\ \mathbf{RF}(Q, \mathit{pr}_1(\mathit{funTSR}(L))).$$

We define $\mathbf{P}(\mathbf{f}, L)$ by: if $L \in \mathbb{PL}$, then, for each $Q \in \mathbb{PL}$, $(\mathbf{RF}(Q, L)\ \&\ \mathbf{ST}(Q)) \Rightarrow \mathbf{RF}(Q, \mathbf{f}(L))$. Since $L$ and $\mathit{rotate}(L)$ share same elements, $\mathbf{RF}(Q, \mathit{rotate}(L))$ follows from $\mathbf{RF}(Q, L)$. So, we have (13).

We next show (14). Fix a list $L$. We show that $L$ satisfies (14) by induction on $i \leqq \mathit{lg}(L)$. Fix $i < \mathit{lg}(L)$, set $L_i := \mathbf{sp}\langle i\rangle(L)$, and let $Q$ be a list with $\mathbf{ST}(Q)\ \&\ \mathbf{RF}(Q, L)$. Then, by induction hypothesis, $\mathbf{RF}(Q, L_i)$. If $\mathit{car}(L) \cap \mathit{Pre}(\mathit{car}(L_i)) = \emptyset$ or $\mathit{car}(L) \cap \mathit{Pre}(\mathit{car}(L_i)) = \mathit{car}(L)$, then $\mathit{split1}(\mathit{car}(L))(L_i)) = \mathit{rotate}(L_i)$, and hence, $\mathbf{RF}(Q, \mathit{split1}(\mathit{car}(L))(L_i))$.

Set $B := \mathit{car}(L) \cap \mathit{Pre}(\mathit{car}(L_i))$ and assume that $B \neq \emptyset$ and $B \neq \mathit{car}(L)$. Then, $\mathit{split1}(\mathit{car}(L))(L_i) = \mathit{enq}(\mathit{enq}(\mathit{cdr}(L_i), B), \mathit{car}(L_i) - B)$. Therefore, in order to show that $\mathbf{RF}(Q, \mathit{split1}(\mathit{car}(L))(L_i))$, it suffices to show that, for each $C\ \varepsilon\ Q$,

$$C \cap B \neq \emptyset\ \Rightarrow\ C \subset B; \tag{15}$$

$$C \cap (car(L_i) - B) \neq \emptyset \\ \Rightarrow C \subset (car(L_i) - B). \quad (16)$$

Let $C \ \varepsilon \ Q$ with $C \cap B \neq \emptyset$. Since $\mathbf{RF}(Q, L)$, $C \subset car(L)$ follows from $C \cap car(L) \neq \emptyset$. We show $C \subset Pre(car(L_i))$. Since $\mathbf{RF}(Q, L_i)$, there exist $C_1, C_2, \ldots, C_k \ \varepsilon \ Q$ with $car(L_i) = \bigcup_{j \leqq k} C_j$. So, since $Pre(car(L_i)) = \bigcup_{j \leqq k} Pre(C_j)$, $C \cap Pre(car(L_i)) \neq \emptyset$ implies that there exists $j_0 \leqq k$ with $C \cap Pre(C_{j_0}) \neq \emptyset$. Therefore, since $Q$ is stable, $C \subset Pre(C_{j_0})$, and hence, $C \subset Pre(car(L_i))$.

Next we show (16). Since $car(L_i) - B = (car(L_i) - car(L)) \cup (car(L_i) - Pre(car(L_i)))$, it suffice to show that, for each $C \ \varepsilon \ Q$,

$$C \cap (car(L_i) - car(L)) \neq \emptyset \\ \Rightarrow \ C \subset (car(L_i) - car(L)); \quad (17)$$

$$C \cap (car(L_i) - Pre(car(L_i))) \neq \emptyset \\ \Rightarrow \ C \subset (car(L_i) - Pre(car(L_i))). \quad (18)$$

Assume that $C \cap (car(L_i) - car(L)) \neq \emptyset$. Since $\mathbf{RF}(L_i, L)$, $car(L_i) - car(L) \neq \emptyset$ means that $car(L_i) - car(L) = car(L_i)$. So, we have $C \subset (car(L_i) - car(L))$ since $\mathbf{RF}(Q, L_i)$.

Next assume that $C \cap (car(L_i) - Pre(car(L_i))) \neq \emptyset$. It will be sufficient to show that $C \cap Pre(car(L_i)) = \emptyset$ from $C \not\subset Pre(car(L_i))$. Suppose that $C \cap Pre(car(L_i)) \neq \emptyset$. Then, we can obtain some $D \ \varepsilon \ Q$ with $D \subset car(L_i)$ and $C \cap Pre(D) \neq \emptyset$ by the same way as that in the proof of (15). So, since $Q$ is stable, $C \subset Pre(D) \subset Pre(car(L_i))$, which contradicts $C \not\subset Pre(car(L_i))$.

So, we have (18), hence (11), and hence (12) for each list $L$. Therefore, since $\mathbf{P}(id, L)$ for each list $L$, we have (PC4).

# 6 PVS-file for the verification of the correctness of TSR

In this section, we briefly explain the content of the PVS-file named `tsr.pvs`, which is available at the website [7]. We use `typewriter` fonts for entities in the pvs-files.

`tsr.pvs` consists of fourteen theory parts we will explain in this section. Almost all theory parts have a parameter `X` which means the set of states of a transition system, while several theory parts have a parameter `R` which means a transition relation. Note that the parameter `X` is assumed to be non-empty and finite (as a set) in many theory parts.

**Remark 6.1** The name of each lemma in each theory part are denoted by `le_`(a list of numbers)`_`(keyword) such as `le_3_1_1_stability`. For two lemmas `le_`$s$`_name` and `le_`$t$`_name`, if $s$ is a initial segment of $t$, then `le_`$t$`_name`

is a lemma used to show `le_`$s\_name$. For example, `le_5_1_1_stability` and `le_5_1_2_ stability` are sub-lemmas of the lemma `le_5_1_stability`. Moreover, `le_5_1_1_1_stability` is a sub-lemma of `le_5_1_1_stability`.

We itemize all theory parts in `tsr.pvs`, as follows.

1. `preliminary_0`

   This theory part consist of the definition of *iter* we explained in Section 3.2 and several lemmas for elementary properties of sets and so on. We make them to simplify the verification in PVS.

2. `product`

   This part has only one lemma assuring that, for each $p \in [X, Y]$, $p = (pr_1(p), pr_1(p))$, where $X$ and $Y$ are parameters of this theory part meaning sets. As well as `preliminary_0`, this part is made only to simplify the verification in PVS. The reason why this lemma is separated from `preliminary_0` is because we want this lemma to have two parameters $X$ and $Y$.

3. `preliminary_1`

   In this theory part, we define *Pre*, *enq* and *rotate* as well as show fundamental properties of lists.

4. `block_and_partition`

   We here define the set of partition lists by defining a type `PTLIST[X]`.

5. `definition_of_TSR`

   Here we define *split1*, *step* and *funTSR* we explained in Section 3.2.

6. `weight_of_partition`

   In this theory part, we define the functional $W$ in Section 4.3. In the actual work of defining $W$, we employ `ordinal` directly without defining *Od* explained in Sections 4.2 and 4.3 (cf. Section 4.4).

   More strictly, we first define a functional `weight` in a slightly differ way from Definition 4.3, and then show that `weight` is (essentially) same as $W$ by a lemma (cf. the definition of `weight` and the lemma `le_2_weight`). The reason why we define `weight` ($= W$) in such a way is because we can simplify the actual work to check well-definedness of `weight`.

7. `property_of_weight`

   We here describe elementary properties of $W$ in order to show some lemmas in Section 4. For example, there are the following lemmas:

   - `le_1_enq_weight`, which correspond to Lemma 4.5.3;
   - `le_2_enq_weight`, which corresponds to Lemma 4.5.2.

8. `property_of_list_operation`

   We here describe elementary properties of lists, in particular, properties of *rotate*. These properties are used to show the correctness of TSR.

9. `termination_of_TSR`

   We here formalize the proof of termination of TSR according to the way in Section 4.3.

10. `meta_property_of_circ`

    This theory part consists of lemmas for *rotate*. We made this theory part in order to simplify the verification of the partial correctness of TSR.

11. `correctness_of_TSR_1`

    We here establish a formal proof of (PC1) according to a similar way to that we explained in Section 5.2.

12. `correctness_of_TSR_2`

    We here establish a formal proof of (PC2) according to a similar way to that in `correctness_of_TSR_1`.

13. `correctness_of_TSR_3`

    We here establish a formal proof of (PC3) according to Section 5.1.

    Since this theory part is slightly more complicated than others, we itemize correspondences of several lemmas in `correctness_of_TSR_3` to lemmas in Section 5.1, as follows.

| | | |
|---|---|---|
| `le_2_stability` | $\Leftrightarrow$ | Lemma 5.1 |
| `le_3_stability` | $\Leftrightarrow$ | Lemma 5.2 |
| `le_4_stability` | $\Leftrightarrow$ | Lemma 5.3 |
| `le_5_1_stability` | $\Leftrightarrow$ | Lemma 5.4 |
| `le_5_stability` | $\Leftrightarrow$ | Lemma 5.5 |
| `le_6_stability` | $\Leftrightarrow$ | (4) and (5) in Section 5.1 |
| `le_7_stability` | $\Leftrightarrow$ | (10) in Sec.5.1 |

    The proof of (PC3) described after Lemma 5.5 is formalized as the proof of `thm_stability`.

14. `correctness_of_TSR_4`

    We here establish a formal proof of (PC4) according to Section 5.2.

# 7 Formal verification of TSR as a more abstract concept

In this section, we consider a formal verification of TSR more abstractly [4]. The main purpose of this paper is to verify TSR including the algorithm how unstable partitions are stabilized. However, if one verify TSR at a more absract level without considering such an algorithm, he can verify TSR much more easily.

We first take a non-empty set $X$ and a transitive order $<$ on $X$ in order to describe that $x$ is a refinement of $y$ by $x \leqq y$. We also take a function $f$ on $X$ satisfying that

$$\forall x \in X \quad f(x) \leqq x. \tag{19}$$

By (19) one can consider $f$ as a kind of "stabilizing" function.

We assume that

$$\forall x, y \in X \ \ (y \leqq x \ \& \ y = f(y) \ \Rightarrow \ y \leqq f(x)) \, .$$

Moreover, we assume that $<$ is well-founded in order to assure that one can not infinitely iterate making proper refinements from any element $x$

$$x > x' > x'' > \cdots \, .$$

We consider TSR as a function $X \to X$ which iterates applying $f$ to a given $x \in X$ until $x$ is stable, i.e., $f(x) = x$. So, we formalize TSR *fix* ($:= \mathit{fix}[f]$) such that TSR returns a fixed point of $f$.

$\mathit{fix} : X \to X$
$\mathit{fix}(x) =$ IF $(x = f(x))$ THEN $x$
ELSE $\mathit{fix}(f(x))$

Next, in terms of *fix*, we consider correctness of TSR corresponding to (TM), (PC2)∼(PC4) in Section 3.2, as follows.

- *fix* is a total function [5].
- For each $x \in X$, $x \geqq \mathit{fix}(x)$.
- For each $x \in X$, $f(\mathit{fix}(x)) = \mathit{fix}(x)$.
- For each $x, y \in X$, if $y \leqq x$ and $y = f(y)$, then $f(y) \leqq \mathit{fix}(x)$.

These properties can be easily shown by using induction on $<$.

The pvs-files `ab_tsr.pvs` and `ab_tsr.prf` concerned with this verification can be downloaded in [7].

[4] We wish to thank a referee for his suggestion which give us an opportunity to consider this verification.

[5] In PVS, the totality of *fix* is checked in the type-checking of *fix*. So, we need not to describe the totality of *fix* explicitly as a formalized proposition in PVS, and we can easily check it by assumptions of $<$ and $f$.

# 8 Conclusion and future work

In this paper, we formalized the algorithm TSR as a functional of lists as well as termination and partial correctness of TSR in the proof assistance PVS.

We formally proved termination of TSR, based on well-foundedness of $[\mathbb{L}, \mathbb{N}]$, which is the domain as well as the codomain of *step*. We also proved partial correctness of TSR.

Finally, we mention some future works subsequent to this paper. The first future work will be to establish formal verifications of more sophisticated minimizing algorithms such as those in [3], [5], [1] and [4]. The second one will be to consider several algorithms dealing with more complicated mathematical structures such as trees or graphs and to verify such algorithms. Formal verifications of them will need more complicated well-founded order, and we will attempt to establish more strong ordinal notation systems to define such orders and to prove the well-foundedness of them.

**Acknowledgments.** We would like to thank the referees for their careful reading and helpful advices. This work was partially supported by Special Coordination Funds for Promoting Science and Technology of Ministry of Education, Culture, Sports, Science and Technology.

# References

[1] Bouajjani, A., Fernandez, J-C. and Halbwachs, N.: Minimal Model Generation, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Volume 3, 1991, pp. 85–91.

[2] Clarke, Jr. E. M., Brumberg, O. and Peled, D. A.,: *Model Checking*, MIT Press 1999.

[3] Kanellakis, P. and Smolka, S.: CCS expressions, finite state processes and thee problems of equivalence, in *Proc. ACM Symposium on Principles of Distributed Computing*, 1983, pp. 228–240.

[4] Lee, D. and Yannakakis, M.: On-line Minimization of Transition Systems, in *Proc. 24th Annual ACM Symposium on Theory of Computing*, May 1992, pp. 264–274

[5] Paige, R. and Tarjan, R. E.: Three partition refinement algorithms, *SIAM J. Comput.*, Vol. 16, No. 6, 1987, pp. 973–989.

[6] Web site of SRI Computer Science Laboratory (which develops PVS): http://pvs.csl.sri.com/

[7] A web-page of H.Watanabe's web-cite: `http://staff.aist.go.jp/hiroshi-watanabe/takaki/` where one can download the pvs-file "`tsr.pvs`" which we mention in this paper as well as a prf-file "`tsr.prf`" and a strategy-file.

Verification of Transition System Reduction via PVS
(算譜科学研究速報)
発行日：2005 年 2 月 25 日
編集・発行：独立行政法人産業技術総合研究所関西センター尼崎事業所
システム検証研究センター
同連絡先：〒661-0974 兵庫県尼崎市若王寺 3-11-46
e-mail：informatics-inquiry@m.aist.go.jp


Verification of Transition System Reduction via PVS
(Programming Science Technical Report)
February 25, 2005
Research Center for Verification and Semantics (CVS)
AIST Kansai, Amagasaki Site
National Institute of Advanced Industrial Science and Technology (AIST)
3-11-46 Nakouji, Amagasaki, Hyogo, 661-0974, Japan
e-mail: informatics- inquiry@m.aist.go.jp