Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2017,7
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Some Notes on Permutations

Peter H. Schmitt

2017

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Fakultät für **Informatik**

Some Notes on Permutations

Peter H. Schmit

Karlsruhe Institute of Technology (KIT), Dept. of Informatics
Am Fasanengarten 5, 76131 Karlsruhe, Germany

Abstract. This note states and proves a theorem on permutations that solves a problem that turned up during the verification of a Java program implementing the dual pivot quicksort algorithm.

1 Introduction

During an attempt to formally verify the dual Quicksort algorithm implemented in JDK the question arose whether the conjecture, now formulated as Theorem 1 below, is true. And if it is how it could be proved, or even formally proved. These notes answer these questions.

The conjecture is in fact true. Lemma 4 plays the crucial role in the proofs. In fact, the proof of the theorem consists merely in reducing its claim to the statement of the lemma. Essential for the proofs of the lemma is the property of a function to be s-stabilizing for a finite sequence s and the property of a function to be $\mathcal{P}$-stabilizing for a partition $\mathcal{P}$ of an initial segment of the natural numbers. The properties are closely related, one might say they are two sides of the same coin. The first proof of Lemma 4 is based on the s-stabilizing property, the second on the $\mathcal{P}$-stabilizing property. In this second approach the claim of the lemma becomes very transparent, also most trivial.

The proof for both Lemma 4 and Theorem 1 have been checked by using KeY as an interactive automated proof system. Appendix A contains the formulation of these results in KeY's language for formulating proof rules while Appendix B and Appendix C contain the corresponding proof trace respectively the proof script.

2 Notation

Definition 1. *A permutation $\rho : X \to X$ of a set X is a surjective and injective function.*
We will use $\mathcal{S}(X)$ to denote the set of permutations on X.

For each natural number $n \in \mathbb{N}$, $n > 0$ we use $\mathcal{S}(n)$ for the special case $\mathcal{S}(\{0, \ldots, n-1\})$.

Definition 2. *Let $s, t : n \to W$ be two finite sequences of length n and range W.*
We say that s is a permutation of t if there is a permuation $\sigma \in \mathcal{S}(n)$ such that $t[i] = s[\sigma(i)]$ for all $0 \leq i < n$. Here, σ is called a witness.

We trust that the use of the same word *permutation* with two different meanings does not cause any problems since the first meaning is a unary predicate while the second is a binary relation.

We may view a permutation $\rho \in \mathcal{S}(n)$. as any other functions, as the set of pairs $\{(i, \rho(i)) \mid 0 \leq i < n\}$. This allows us to use set theoretic operations on permutations. We may thus speak of the union of two permutation $\rho_1 \cup \rho_2$, of ρ_1 being a subset of ρ_2, $\rho_1 \subseteq \rho_2$, etc. In general the union of two permutation is not a permutation, not even a function. But

Lemma 1. *Let ρ_1 be a permutation of the set $X \subseteq \mathbb{N}$ and ρ_2 a permutation of the set $Y \subseteq \mathbb{N}$ with X and Y disjoint then $\rho_1 \cup \rho_2$ is a permutation of $X \cup Y$.*

Proof. Obvious. □

Definition 3. *Let $s : n \to W$ be a finite sequence.*
A function (partial function) $f : \{0, \ldots n-1\} \to \{0, \ldots n-1\}$ is called stablizing for s if for all $0 \leq i < n$ (for all i in the domain of definition of f) we have

$$s[i] = s[f(i)].$$

By $Stab(s)$ we denote the set of all stabilizing permuations for s.

If s is injective, i.e., $i \neq j$ implies $s[i] \neq s[j]$, then of course $Stab(s)$ consists only of the identity permuations, $Stab(s) = \{id\}$.

Definition 4. *Let $\mathcal{P} = (P_i)_{0 \leq i < k}$ be a partition of $n = \{0, \ldots, n-1\}$.*
A function (partial function) $f : \{0, \ldots n-1\} \to \{0, \ldots n-1\}$ is called stablizing for $\mathcal{P}$ if for all $0 \leq i < n$ an all $x \in \{0, \ldots n-1\}$ (all x in the e domain of definition of f)

$$x \in P_i \text{ implies } f(x) \in P_i.$$

We denote the $\mathcal{P}$-stablizing permutations by $Stab(\mathcal{P})$.

Definitions 1 and 4 are closely related.

Lemma 2.

1. *Let $s : n \to W$ be a finite sequence and $f : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ a function (partial function).*
   If f is stabilizing for s the f is also $\mathcal{P}$-stabilizing for $\mathcal{P} = (P_i)_{0 \leq i < k}$ with $W = \{w_i \mid 0 \leq i < k\}$ and $P_i = \{m < n \mid f(m) = i\}$. We omit possibly empty sets P_i from the partition.
2. *Let $\mathcal{P} = (P_i)_{0 \leq i < k}$ be a partition of $n = \{0, \ldots, n-1\}$ and $f : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ a function (partial function).*
   If f is stabilizing for $\mathcal{P}$ then f is also s-stabilizing or the sequence $s : n \to W$ with $W = \{0, \ldots, k-1\}$ and $s(m) = i$ iff $m \in P_i$ for all $m < n$.

Proof. The definition of $\mathcal{P}$ from s ins case 1 and the definition of s from $\mathcal{P}$ in case 2 satisfy for all $x, y \in \{0, \ldots, n-1\}$ the following equivalence

$$s[x] = s[y] \Leftrightarrow \text{ for all } i \in \{0, \ldots, k-1\}(x \in P_i \to y \in P_i)$$

□

3 The Results

Lemma 3. *Let $s : n \to W$ be a permutation of t with witness σ. Furthermore let $\rho \in Stab(s)$.*
Then $\rho\sigma$ is also a witness of the permutation.

Proof. For $0 \leq i < n$ we obtain

$$s[\rho\sigma(i)] = s[\rho(\sigma(i))] = s[\sigma(i)] = t[i]$$

Lemma 4. *Let $s : n \to W$ be a finite sequence.*
For every partial injective s-stabilizing function $\rho_0 : n \to n$ there is $\rho \in Stab(s)$ extending ρ_0.
In greater detail:
Let X, Y be subsets of $\{0, \ldots, n-1\}$ and $\rho_0 : X \to Y$ a bijection from X onto Y such that $\forall x \in X(s[x] = s[\rho_0(x)])$.
Then there is a permutation ρ in $Stab(s)$ that extends ρ_0, i.e., $\rho_0 \subseteq \rho$.

Proof.

$$Y = Y_0 \cup Y_1 \quad \text{with } Y_0 = Y \cap X \text{ and } X \cap Y_1 = \emptyset \tag{1}$$

To extend ρ_0 to a permutation $\rho \in \mathcal{S}(n)$ we need to find for every $0 \leq i < n$ that is not in X a value $\rho(i)$ that is not in Y.
For $0 \leq i < n$ with $i \notin X$ and $i \notin Y$ we set $\rho(i) = i$.
It remains to deal with $i \in Y_1$. For every $i \in Y_1$ we define a sequence $x_k(i)$ with $x_0(i) = i$ and $\rho_0(x_{k+1}(i) = x_k(i)$. Note, $x_{k+1}(i)$, if it exits, is uniquely determined by $x_k(i)$ by the injectivity of ρ_0. All elements of this sequence are different. $x_k(i)$ for $k > 0$ and $x_0(i) = i$ cannot be equal since they are elements of disjoint sets, $i \in Y_1$ and $x_k(i) \in X$. If $a = x_k(i) = x_m(i)$ with $0 < k < m$ then $\rho_0(a) = x_{k-1}(i)$ and $\rho_0(a) = x_{m-1}(i)$ which is impossible. By this and the fact that all $x_k(i)$ are natural numbers $< n$ each sequence must terminate. $i = x_0(i), x_1(i), \ldots, x_{k(i)}(i)$ with $x_{k(i)}(i) \notin Y$. We complete the definition of ρ by

$$\rho(i) = x_{k(i)}(i) \quad \text{for } i \in Y_1 \tag{2}$$

It is easily checked that $\rho \in \mathcal{S}(n)$. By the assumptions on ρ_0 we also know $s[x_{k(i)}(i)] = s[i]$ Thus $\rho \in Stab(s)$. $\square$

Theorem 1. *Let $s, t : n \to W$ be finite sequences and s a permutation of t. Consider furthermore $X_0, Y \subseteq \{0, \ldots, n-1\}$ and σ_0 a bijective mapping from X_0 onto Y such that*

$$t[x] = s[\sigma_0(x)] \quad \text{for all } x \in X_0$$

Then there is a permuation $\sigma \in \mathcal{S}(n)$

1. *extending σ_0, i.e., $\sigma \downarrow X_0 = \sigma_0$ and*

2. $t[i] = s[\sigma(i)]$ *for all* $0 \leq i < n$
   i.e., σ *is a witness of* s *being a permutation of* t*.*

Proof. Since s is a permutation of t there is $\sigma' \in \mathcal{S}(n)$ with

$$t[i] = s[\sigma'(i)] \quad \text{for all } 0 \leq i < n \tag{3}$$

Set $X = \sigma'(X_0)$ and define $\rho_0 : X \to Y$ by $\rho_0 = \sigma_0 \circ (\sigma')^{-1}$. Obviously, ρ_0 is a bijection from X anto Y. From the properties of σ' and σ_0 we derive

$$s[x] = s[\rho_0(x)] \quad \text{for all } x \in X$$

Thus Lemma 4 is applicable and we obtain $\rho \in Stab(s)$ extending ρ_0. By Lemma 3 also $\sigma = \rho \circ \sigma'$ is a witness for s being a permutation of t. In addition, we obtain for all $x \in X_0$

$$\begin{aligned} \sigma(x) &= \rho(\sigma'(x)) \\ &= \sigma_0((\sigma')^{-1}(\sigma'(x))) \\ &= \sigma_0(x) \end{aligned}$$

□

Corollary 1. *Let* $s : n \to W$ *be a permutation of* t *and* $0 \leq x, y < n$ *indices with*

$$t[x] = s[x] \tag{4}$$

and

$$t[y] = s[y] \tag{5}$$

.
Then there is a witness σ *such that* $\sigma(x) = x$ *and* $\sigma(y) = y$*.*

Proof. Follows from Theorem 1 by $X_0 = Y = \{x, y\}$ and $\rho_0(x) = x$, $\rho_0(y) = y$.
□

Alternative Proof of Corollary 1
This alternative proof is unrelated to the rest of these notes. It is a kind of finger exercise. The intention was to come up with a proof plan that is not deep but wide, i.e., that consists of many simple case, 12 cases in fact. The hope was that such a proof plan would be more amenable for an automated reasoning system. This was however never put to the test.

The idea of this alternative proof is to specialize the combined proofs of Lemma 4 and Theorem 1 to the very special situation of the corollary.
So we start with $X_0 = \{x, y\}$, $\sigma_0 : X_0 \to X_0$ is the identity function, thus $Y = X_0$ and $\sigma' \in \mathcal{S}(n)$ witnessing that s is a permutation of t. $X = \{\sigma'(x), \sigma'(y)\}$. Next, $\rho_0 : \{\sigma'(x), \sigma'(y)\} \to \{x, y\}$ is given by $\rho_0(\sigma'(x)) = x$ and $\rho_0(\sigma'(y)) = y$.

Case A: $x = y$

Case A1: $\sigma'(x) = x$ Nothing to do. We can use σ' for σ.

Case A2: $\sigma'(x) \neq x$ Let σ be σ' followed by a swap of $\sigma'(x)$ and x. Note, that σ can also be represented as $\sigma = seqSwap(\sigma', x, u)$ where $\sigma'(u) = x$.

Case B: $x \neq y$ The case assumption implies $\sigma'(x) \neq \sigma'(y)$.

Case B1: $\sigma'(x) = x$ *and* $\sigma'(y) = y$ Nothing to do. We can use σ' for σ.

Case B2: $\sigma'(x) = x$ *and* $\sigma'(y) \neq y$ We obtain $Y_0 = \{x\}$ and $Y_1 = \{y\}$. Note, that $y = \sigma'(x) = x$ is not possible, it would imply the contradiction to the case assumption $x \neq y$. Following the definition in the proof of Lemma 4 we obtain $x_{k(y)}(y) = \sigma'(y)$. Unravelling these definitions we see that we may define σ to be σ' followed by a swap of $\sigma'(y)$ and y. Note, $\sigma = seqSwap(\sigma', y, w)$ with $\sigma'(w) = y$.

Case B3: $\sigma'(x) \neq x$ *and* $\sigma'(y) = y$ This is symmetric to case B2 and we obtain σ to be σ' followed by a swap of $\sigma'(x)$ and x.

Case B4: $\sigma'(x) \neq x$ *and* $\sigma'(y) \neq y$ In all the following cases these two conditions are implicitely assumed.

Case B4i: $\sigma'(x) = y$ *and* $\sigma'(y) \neq x$ $Y_0 = \{y\}$, $Y_1 = \{x\}$. $x_0(x) = x$, $x_1(x) = \sigma'(x) = y$, $x_2(x) = \sigma'(y)$ and $k(x) = 2$. Thus ρ is given by $\rho(\sigma'(x)) = x$, $\rho(\sigma'(y)) = y$, $\rho(x) = \sigma'(y)$ and $\rho(i) = i$ for all other i. Furthermore, $\sigma = \rho \circ \sigma'$ or explicitly

$$\sigma(i) = \begin{cases} x & \text{if } i = x \\ y & \text{if } i = y \\ \sigma'(y) & \text{if } \sigma'(i) = x \\ \sigma'(i) & \text{otherwise} \end{cases}$$

Careful inspection shows that $\sigma = seqSwap(seqSwap(\sigma', u, x), u, y)$ with, as usual, $\sigma'(u) = x$.

Case B4ii: $\sigma'(x) \neq y$ *and* $\sigma'(y) = x$ This is symmetric to case *B4i i.e., we may use* σ *with*

$$\sigma(i) = \begin{cases} x & \textit{if } i = x \\ y & \textit{if } i = y \\ \sigma'(x) & \textit{if } \sigma'(i) = y \\ \sigma'(i) & \textit{otherwise} \end{cases}$$

or $\sigma = seqSwap(seqSwap(\sigma', w, y), w, x)$ *with, as usual,* $\sigma'(w) = y$.

Case B4iii: $\sigma'(x) = y$ *and* $\sigma'(y) = x$ *In this case we get from* $\sigma'(y) = x$ *and* $\sigma'(u) = x$ *the equalitiy* $u = y$. *The permuation from* B4i reduces to $\sigma = seqSwap(seqSwap(\sigma', u, x), y, y) = (seqSwap(\sigma', u, x)$.

Case B4iv: $\sigma'(x) \neq y$ and $\sigma'(y) \neq x$ $Y_0 = \emptyset$, $Y_1 = \{x, y\}$. Futhermore, $x_1(x) = \sigma'(x)$, $k(x) = 1$ and $x_1(y) = \sigma'(y)$, $k(y) = 1$. Thus σ is obtained by σ' and the two swaps $\sigma'(x)$ with x and $\sigma'(y)$ with y. Or explicitely,

$$\sigma(i) = \begin{cases} x & \text{if } i = x \\ \sigma'(x) & \text{if } \sigma'(i) = x \\ y & \text{if } i = y \\ \sigma'(y) & \text{if } \sigma'(i) = y \\ \sigma'(i) & \text{otherwise} \end{cases}$$

or $\sigma = seqSwap(seqSwap(\sigma', x, u), y, w)$ with $\sigma'(u) = x$ and $\sigma'(w) = y$.
In all cases we have to convince ourselves that σ is indeed a witness for s being a permutation of t. But, once we know what σ should be, this is easy. □

Corollary 2. *Let $s : n \to W$ be a permutation of t and $X_0 \subseteq \{0, \ldots, n-1\}$ such that $s[x] = t[x]$ for all $x \in X_0$.*
Then there is a witness σ such that $\sigma(x) = x$ all $x \in X_0$.

Proof. Follows from Theorem 1 with $\rho_0(x) = x$ for all $x \in X_0$. □

4 A Second Approach

Lemma 5. *Let $\mathcal{P} = (P_i)_{0 \leq i < k}$ be a partition of $n = \{0, \ldots, n-1\}$ and $f : n \to n$ a function (partial function).*
Then f is stabilizing for $\mathcal{P}$ iff f is the disjoint union of functions (partial functions) $f_i : P_i \to P_i$ for $0 \leq i < k$, in symbols: $f = \bigcup_{0 \leq i < k} f_i$

Proof. Define for $0 \leq i < k$ the function f_i with P_i (Pi intersected with the domain of definition of f in the partial function case) as it sdomain of definition by

$$f_i(x) = f(x)$$

Since $\mathcal{P}$ is a partition we get in any case

$$f = \bigcup_{0 \leq i < k} f_i$$

If f is stabilizing for $\mathcal{P}$ then we see that the range of f_i is contained in P_i.
On the other hand of we know that the range of f_i is contained in P_i then the union is $\mathcal{P}$-stabilizing. □

Second Proof of Lemma 4

Proof. By Lemma 2(1) ρ_0 is $\mathcal{P}$-stabilizing for the partition $\mathcal{P} = (P_i)_{0 \leq i < k}$ of $\{0, \ldots n-1\}$ defined there. By Lemma 5 we can write $\rho_0 = \bigcup_{0 \leq i < k} \rho_0^i$ for partial functions $\rho_0^i : P_i \to P_i$. Since ρ_0 was assumed to be injective all ρ_0^i are also injective partial functions. It is easy to extend each ρ_0^i to a $\rho^i \in \mathcal{S}(P_i)$. Then $\rho = \bigcup_{0 \leq i < k} \rho^i$ is an extension of ρ_0 and again by Lemma 5 ρ is $\mathcal{P}$-stabilizing. An appeal to Lemma 2(2) shows that ρ is also s-stabilizing. Note: if $\mathcal{P}$ is defined from s as described in Lemma 2(1) and subsequently s' is defined from $\mathcal{P}$ as described in 2(2) then $s' = s$.

For the readers who want more detail on the extension argument, let $X \subseteq P_i$ be the domain and $Y \subseteq P_i$ the range partial of the injective function ρ_0^i, i.e., $\rho_0^i : X \to Y$. By injectivity $card(X) = card(Y)$, in other words X and Y have the same number of elements. Thus $card(\{0, \ldots, n-1\} \setminus X) = card(\{0, \ldots, n-1\} \setminus Y)$, i.e. there is a bijection $\rho_1^i : \{0, \ldots, n-1\} \setminus X \to \{0, \ldots, n-1\} \setminus Y$. Now, $\rho^i = \rho_0^i \cup \rho_1^i$ the the permutation of P_i that we were looking for. $\square$

References

1. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive Software Verification - The KeY Book - From Theory to Practice, Lecture Notes in Computer Science, vol. 10001. Springer (2016), http://dx.doi.org/10.1007/978-3-319-49812-6

A Taclets

```
lemma
  schiffl_lemma_2 {
    \schemaVar \term Seq s, t;
    \schemaVar \variable  Seq r;
    \schemaVar \variable  int x, y, iv;

    \find (seqPerm(s,t)==>)
     \varcond (\notFreeIn (iv,s,t),
               \notFreeIn (r ,s,t),
               \notFreeIn (x ,s,t),
               \notFreeIn (y ,s,t))

    \add(\forall x;\forall y;(
         any::seqGet(s,x)=any::seqGet(t,x) &
         any::seqGet(s,y)=any::seqGet(t,y) & 0 <= x & x < seqLen(s) &
         0 <= y & y < seqLen(s)
      -> \exists r; (seqLen(r) = seqLen(s) & seqNPerm(r) &
           (\forall iv; (0 <= iv &  iv < seqLen(s) ->
            any::seqGet(s,iv) = any::seqGet(t,int::seqGet(r,iv)))) &
            int::seqGet(r,x)= x &  int::seqGet(r,y)= y))
      ==> )
  };
```

Fig. 1. Taclet for Lemma 4

To use Lemma 4 and Theorem 1 in the KeY system they have to be formulated in KeY's language for proof rules. Rules in this format are called *taclets*. Full explanation of the taclet language can be found in [1, Chapter 4].

Figure 1 shows the taclet for Lemma 4. It has been proved with the KeY prover with the proof script reproduced in Appendix B.

nodes	8819
branches	135
quantifier instantiations	42
One-step simplifications	123
totel rule applications	8950

Figure 2 shows the taclet for Theorem 1. It has been proved with the KeY prover with the proof script reproduced in Appendix C

```
 \lemma
  schiffl_thm_1 {
    \schemaVar \term Seq s, t;
    \schemaVar \term int x, y;
    \schemaVar \term any a, b;
    \schemaVar \variables int idx;
    \varcond (\notFreeIn(idx, x),  \notFreeIn(idx, y),
              \notFreeIn(idx, a),  \notFreeIn(idx, b),
              \notFreeIn(idx, s),  \notFreeIn(idx, t))

    \add(seqPerm(s,t) & any::seqGet(s,x)=any::seqGet(t,x) &
         any::seqGet(s,y)=any::seqGet(t,y) & 0 <= x & x < seqLen(s) &
         0 <= y & y < seqLen(s) ->
seqPerm(seqDef{idx;}(0,s.length,\if(idx=y)\then(b)\else
                                   (\if(idx=x)\then(a)\else(any::seqGet(s, idx))))
       ,seqDef{idx;}(0,s.length,\if(idx=y)\then(b)\else(
                                   \if(idx=x)\then(a)\else(any::seqGet(t, idx)))))
==> )
  };
```

Fig. 2. Taclet for Theorem 1

nodes	830
branches	17
quantifier instantiations	11
One-step simplifications	30
totel rule applications	850

B Proof Script for Lemma 4

```
\profile "Java Profile";

\settings {
  omitted
}

\proofObligation "#Proof Obligation Settings
#Thu Oct 27 13:34:28 CEST 2016
name=schiffl_lemma_2
class=de.uka.ilkd.key.taclettranslation.lemma.TacletProofObligationInput
";

\proofScript "
macro split-prop;
rule allRight;
rule allRight;
macro split-prop;
rule 'seqPermDefLeft';
rule 'andLeft';
rule 'exLeft';
macro split-prop;
# the following equations are useful in many case.
rule  seqNPermRange;
instantiate var=iv with='v_x_0' occ=1;
rule impLeft;
tryclose branch;
rule andLeft;
rule andLeft;
# 1. triple of equations
instantiate var=iv with='v_y_0' occ=1;
rule impLeft;
tryclose branch;
rule andLeft;
rule andLeft;
# 2. triple of equations
rule seqNPermDefLeft;
instantiate var=iv with='v_x_0' occ=2;
rule impLeft;
tryclose branch;
rule exLeft;
rule andLeft;
rule andLeft;
# 3. triple of equations
instantiate hide var=iv with='v_y_0' occ=2;
```

```
rule impLeft;
tryclose branch;
rule exLeft;
rule andLeft;
rule andLeft;
# 4. triple of equations
instantiate var=iv with='jv_0' occ=1;
rule impLeft;
tryclose branch;
rule andLeft;
rule andLeft;
rule castAdd formula='s_0[jv_0] = v_x_0' occ=0;
# 5. set of equations
instantiate hide var=iv with='jv_1' occ=1;
rule impLeft;
tryclose branch;
rule andLeft;
rule andLeft;
rule castAdd formula='s_0[jv_1] = v_y_0' occ=0;
# 6. set of equations
instantiate var=iv  with='v_x_0';
rule impLeft;
tryclose branch;
# 7. equation
instantiate var=iv  with='v_y_0';
rule impLeft;
tryclose branch;
# 8. equation
cut 'v_x_0 = v_y_0';
# This corresponds to case A in the Notes.
instantiate hide var='v_r' with='seqSwap(s_0,v_x_0,jv_0)';
# in the following r refers to this instantion
rule andRight;
rule andRight;
rule andRight;
rule andRight;
rule lenOfSwap;
tryclose branch;
# established: r is of correct length
rule seqNPermSwapNPerm;
instantiate hide var='iv' with='v_x_0' occ=1;
instantiate hide var='jv' with='jv_0';
rule impLeft;
tryclose branch;
tryclose branch;
```

```
# established: r is permutation
rule allRight;
rule impRight;
rule andLeft;
instantiate var=iv with='v_iv_0';
rule impLeft;
tryclose branch;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
#  established:  witness property of r
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
# established: r fixes v_x_0
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
# established: r fixes v_y_0
# from now on v_x_0 != v_y_0
cut '(int)s_0[v_x_0] = (int)s_0[v_y_0]';
rule seqNPermInjective;
```

```
instantiate hide var=iv with='v_x_0';
instantiate hide var=jv with='v_y_0';
rule impLeft;
tryclose branch;
tryclose branch;
## from now on s_0[v_x_0] != v_x_0
cut '(int)s_0[v_x_0] = v_x_0';
# This corresponds to case B1 & B2  in the Notes.
instantiate hide var=v_r with='seqSwap(s_0,v_y_0,jv_1)';
# in the following r1 refers to this instantion
rule andRight;
rule andRight;
rule andRight;
rule andRight;
tryclose branch;
# established: r1 is of the correct length
rule seqNPermSwapNPerm;

instantiate hide var=iv with='v_y_0';
instantiate hide var=jv with='jv_1';
tryclose branch;
# established: r1 is permutation
rule allRight;
rule impRight;
rule andLeft;
instantiate var=iv with='v_iv_1';
rule impLeft;
tryclose branch;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
instantiate var=iv with='v_y_0';
rule impLeft;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
# established: witness property for r1
rule getOfSwap;
```

```
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
# established: r1 fixes v_x_0
tryclose branch;
# established: r1 fixes v_y_0
# from now on  v_x_0 != v_y_0 and s_0[v_x_0]!= v_x_0
cut  '(int)s_0[v_y_0] = v_y_0';
# This corresponds to case B3  in the Notes.
instantiate hide var=v_r with='seqSwap(s_0,v_x_0,jv_0)';
# in the following r2 refers to this instantion
rule andRight;
rule andRight;
rule andRight;
rule andRight;
tryclose branch;
# established: r2 is of the correct length
rule seqNPermSwapNPerm;
instantiate hide var=iv with='v_x_0';
instantiate hide var=jv with='jv_0';
rule impLeft;
tryclose branch;
tryclose branch;
# established: r2 is permutation
rule allRight;
rule impRight;
rule andLeft;
instantiate var=iv with='v_iv_2';
rule impLeft;
tryclose branch;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
# established: witness property for r2
rule getOfSwap;
```

```
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
# established: r2 fixes v_x_0
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
# established: r2 fixes v_y_0
# from now on  v_x_0 != v_y_0 and s_0[v_x_0]!= v_x_0 and  s_0[v_y_0]!= v_y_0
cut '(int)s_0[v_x_0]=v_y_0';
# This corresponds to case B4i & B4iii  in the Notes.
instantiate hide var=v_r with='seqSwap(seqSwap(s_0,jv_0,v_x_0),jv_0,v_y_0)';
# in the following r3 refers to this instantion
rule andRight;
rule andRight;
rule andRight;
rule andRight;
tryclose branch;
# established: r3 is of the correct length
rule seqNPermSwapNPerm;
instantiate hide var=iv with='jv_0';
instantiate hide var=jv with='v_x_0';
rule impLeft;
tryclose branch;
rule seqNPermSwapNPerm formula='seqNPerm(seqSwap(s_0, jv_0, v_x_0))';
instantiate hide var=iv with='jv_0';
instantiate hide var=jv with='v_y_0';
rule impLeft;
tryclose branch;
tryclose branch;
# established: r3 is permutation
rule allRight;
rule impRight;
rule andLeft;
# start: providing equation for latter use
#        in many case distinctions
instantiate var=iv with='v_iv_3';
rule impLeft;
tryclose branch;
# end: providing equation for latter use
rule getOfSwap;
rule ifthenelse_negated;
```

```
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
# established: case v_iv_3=jv_0 in the unravelling of r3
rule ifthenelse_split;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
# established: case v_iv_3=v_y_0 in the unravelling of r3
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
# established: witness property for r3
rule getOfSwap;
```

```
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
# established: r3 fixes v_x_0
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
# established: r3 fixes v_y_0
# from now on  v_x_0 != v_y_0 and s_0[v_x_0]!= v_x_0 and
# s_0[v_y_0]!= v_y_0  and s_0[v_x_0]!= v_y_0
cut 'int::seqGet(s_0, v_y_0) = v_x_0';
# This corresponds to case B4ii  in the Notes.
instantiate hide var=v_r with='seqSwap(seqSwap(s_0,jv_1,v_y_0),jv_1,v_x_0)';
# in the following r4 refers to this instantion
rule andRight;
rule andRight;
rule andRight;
rule andRight;
tryclose branch;
# established: r4 is of the correct length
rule seqNPermSwapNPerm;
instantiate hide var=iv with='jv_1';
instantiate hide var=jv with='v_y_0';
rule impLeft;
tryclose branch;
rule seqNPermSwapNPerm formula='seqNPerm(seqSwap(s_0,jv_1,v_y_0))';
instantiate hide var=iv with='jv_1';
instantiate hide var=jv with='v_x_0';
rule impLeft;
tryclose branch;
tryclose branch;
```

```
# established: r4 is permutation
rule allRight;
rule impRight;
rule andLeft;
# start: providing equation for latter use
#        in many case distinctions
instantiate var=iv with='v_iv_4';
rule impLeft;
tryclose branch;
instantiate var=iv with='v_iv_4';
rule impLeft;
tryclose branch;
# end: providing equation for latter use
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
rule ifthenelse_split occ=0;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
```

```
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
# established: witness property for r4
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
# established: r4 fixes v_x_0
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
```

```
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
# established: r4 fixes v_y_0
# from now on  v_x_0 != v_y_0 and s_0[v_x_0]!= v_x_0 and
# s_0[v_y_0]!= v_y_0  and s_0[v_x_0]!= v_y_0 and  s_0[v_y_0]!=v_x_0;
instantiate hide var='v_r' with='seqSwap(seqSwap(s_0,v_x_0,jv_0),v_y_0,jv_1)';
# this corresponds to case B4iv in the Notes
# in the following r5 refers to this instantion
rule andRight;
rule andRight;
rule andRight;
rule andRight;
tryclose branch;
# established: r5 is of the correct length
rule seqNPermSwapNPerm;
instantiate hide var=iv with='v_x_0';
instantiate hide var=jv with='jv_0';
rule impLeft;
tryclose branch;
rule seqNPermSwapNPerm formula='seqNPerm(seqSwap(s_0,v_x_0,jv_0))';
instantiate hide var=iv with='v_y_0';
instantiate hide var=jv with='jv_1';
rule impLeft;
tryclose branch;
tryclose branch;
# established: r5 is permutation
rule allRight;
rule impRight;
instantiate hide var=iv with='v_iv_5';
rule impLeft;
tryclose branch;
rule andLeft;
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
tryclose branch;
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
```

```
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
rule ifthenelse_split occ=0;
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
# established: witness property for r5
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
```

```
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
rule getOfSwap occ=0;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;
rule andLeft;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
rule getOfSwap;
rule ifthenelse_negated;
rule ifthenelse_split occ=0;
rule andLeft;
rule andLeft;

rule andLeft;
```

```
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
tryclose branch;
# established: r5 fixes v_y_0
"
```

C Proof Script for Theorem 1

```
\profile "Java Profile";
\settings {
"#Proof-Settings-Config-File
  omitted
}

\proofObligation "#Proof Obligation Settings
#Thu Oct 27 16:09:31 CEST 2016
name=schiffl_thm_1
class=de.uka.ilkd.key.taclettranslation.lemma.TacletProofObligationInput
";

\proofScript "
macro split-prop;
rule schiffl_lemma_2 formula='seqPerm(f_s, f_t)';
instantiate hide var=x with='f_x';
instantiate hide var=y with='f_y';
rule impLeft;
tryclose branch;
rule exLeft;
macro split-prop;
rule seqPermDef occ=1;
rule andRight;
tryclose branch;
instantiate hide var=s with='r_0';
rule andRight;
tryclose branch;
rule allRight;
rule impRight;
instantiate hide var=iv with='iv_0';
rule impLeft;
tryclose branch;
rule andLeft;
rule seqNPermRange;
instantiate hide var=iv with='iv_0';
rule impLeft;
tryclose branch;
rule andLeft;
rule andLeft;
rule seqNPermRange;
instantiate hide var=iv with='f_x';
rule impLeft;
tryclose branch;
rule andLeft;
```

```
rule andLeft;
rule seqNPermRange;
instantiate hide var=iv with='f_y';
rule impLeft;
tryclose branch;
rule andLeft;
rule andLeft;
rule getOfSeqDef occ=0;
rule getOfSeqDef;
rule ifthenelse_split occ=0;
rule andLeft occ=0;
rule sub_zero_2 occ=0;
rule ifthenelse_split occ=2;
rule andLeft;
rule sub_zero_2 occ=0;
rule add_zero_right occ=0;
rule add_zero_right occ=0;
rule add_zero_right occ=0;
rule add_zero_right occ=0;
rule add_zero_right occ=0;
rule add_zero_right occ=0;
rule ifthenelse_split occ=0;
rule ifthenelse_split occ=0;
tryclose branch;
tryclose branch;
rule ifthenelse_split occ=0;
tryclose branch;
rule ifthenelse_split occ=0;
rule seqNPermInjective;
instantiate hide var=iv with='iv_0';
instantiate hide var=jv with='f_y';
rule impLeft;
tryclose branch;
tryclose branch;
rule ifthenelse_split occ=0;
rule seqNPermInjective;
instantiate hide var=iv with='iv_0';
instantiate hide var=jv with='f_x';
rule impLeft;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
tryclose branch;
"
```