T U M

I N S T I T U T F Ü R I N F O R M A T I K

Implementing the Change of Data Structures with SPECTRUM in the Framework of KORSO Development Graphs

Oscar Slotosch

TUM-I9511
März 1995

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-03-1995-I9511-350/1.-FI

Typescript: ---

Druck: Mathematisches Institut und
Institut für Informatik der
Technischen Universität München

Implementing the Change of Data Structures with SPECTRUM in the Framework of KORSO Development Graphs*

Oscar Slotosch
email:slotosch@informatik.tu-muenchen.de
http://hpbroy3.informatik.tu-muenchen.de/MITARBEITER/slotosch/slotosch.html

Abstract

Data structures are algebraically specified with abstract data types in the specification language SPECTRUM. Structures of specifications are depicted in the Development-Graphs like in the BMFT-Project: KORSO. These graphs contain refinement relations, which have to be proved.

In this paper a general method for changing data structures is demonstrated on the example of sets and sequences. The focus lays on proving the involved refinement relations with the theorem prover Isabelle. A simple method is given to support such proofs.

Contents

*This work was sponsored by the German Ministry of Research and Technology (BMFT) as part of the compound project "KORSO - Korrekte Software".

1 Introduction

The change of data structures invokes several proof obligations in a deductive software development setting. The task of this paper is to perform the change of data structures, i.e. to prove all verification conditions which occur.

The specification language SPECTRUM is chosen, even if the correctness of the construction can be guaranteed (until now) only for a subset of SPECTRUM specifications (using partial algebras with positive conditional equations of first order logic without existential quantifiers). For showing the relations between the involved hierarchical specifications the KORSO-framework is used.

The next section shortly presents the method, elaborated in [Slo95], whereas the following sections carry through the standard example of implementing sets by sequences with this method. In [PBDD94] a similar change of data structure is described, but from a more methodological point of view without giving and proving all verification conditions.

2 Method for the Change of Data Structures

The method is in [Slo95] described and related to other approaches. It performs an semantic SRI-implementation in the classical sense of abstract data types (see [EKMP82]) by defining the new model to be a quotient on a restriction of the concrete model. Subtyping and overloading aspects are avoided by explicitly modelling the signature-morphism with a function abs. This ensures that the model of the constructed specification, which bases on the concrete sort is contained in the class of models of the abstract specification. Since this inclusion is transitive the method for the change of data structures fits well into the framework of deductive software development (see e.g. [BW93]).

The method gives a scheme for the construction, in which the developer has to fill in several details in order to achieve a correct (and provable) refinement between the abstract and the concrete specification.

The goal is to implement an abstract specification $A = \langle\langle S^A, \Omega^A\rangle, E^A\rangle$ (with an abstract sort $as{\in}S^A$) as a *quotient* of a *subalgebra* from a concrete specification: $C = \langle\langle S^C, \Omega^C\rangle, E^C\rangle$, where the concrete specification contains for the as a corresponding $cs{\in}S^C$ and for every $f^A{\in}\Omega^A$ a corresponding[1] $f^\star{\in}\Omega^C$. To simplify notations $\tilde{c}$ will be the subvector of $\bar{c}$ with $c_i : cs \in \tilde{c}$ contains only elements of the corresponding sort.

[1]corresponding means that for every function $f^A_{\bar{s},s}$ means that $f^\star$ has the arity $f^\star_{\varphi(\bar{s}),\varphi(s)}$ where $\varphi(as) = cs$ and $\varphi(x) = x$ for $x{\neq}as$.

2.1 Syntactic Steps

The method consists of extending an (appropriate) concrete specification in the following steps:

1. Add a predicate $p : cs{\rightarrow}Bool$.
2. Add axioms for p which specify the representing-elements.
3. Add a partial function $abs : cs|_p \rightarrow as$.
4. Add the axiom $\delta(abs(s)) = p(s)$ which defines the domain of abs.
5. For every function f^A add the axiom: $p(\tilde{c}){\Rightarrow}f^A(abs(\bar{c})) = abs(f^\star(\bar{c}))$ where $abs(\bar{c})$ is the point-wise application of abs and $p(\tilde{c})$ is the conjunction of $p(c_i)$ for all $c_i{\in}\tilde{c}$.
6. Add a congruence predicate $\equiv : cs|_p \times cs|_p \rightarrow Bool$.
7. Add axioms for $\equiv$ specifying a congruence on $cs|_p$.
8. Add the axiom $p(s){\wedge}p(t){\Rightarrow}(abs(s) = abs(t)) = (s{\equiv}t)$.
9. Add a generator for the new sort: $G_{cs,as} = \{abs\}$.

Steps 1., 3., 4., 5., 6., 8. and 9. are created schematically after the selection of as and cs. In steps 2. and 7. the user has to specify the right implementation decisions. Step 0. is to find and appropriate specification which implements the abstract one has to be done be the user too. A schematic help is the generation of the signature of the corresponding functions, for which the user has to enter only the definitions.

2.2 Proof Obligations

The extended specification has to fulfil the following proof obligations:

1. E^A all axioms of the abstract type
2. For all $f^A{\in}\Omega^A$ prove the invariance $\bar{p}(\tilde{c}){\Rightarrow}p(f^\star(\bar{c}))$.
3. $\equiv$ has to fulfil the congruence axioms.
   - Reflexivity: $s{\equiv}s$ for all $s{\in}cs|_p$
   - Symmetry: $s{\equiv}t \Leftrightarrow t{\equiv}s$ for all $s, t{\in}cs|_p$
   - Transitivity: $s{\equiv}t \wedge t{\equiv}u \Rightarrow s{\equiv}u$ for all $s, t, u{\in}cs|_p$
   - Substituitivity: for all $f^A{\in}\Omega^A$ prove $\bar{c}{\equiv}\bar{d}{\Rightarrow}f^\star(\bar{c}){\equiv}f^\star(\bar{d})$ where $\bar{c}{\equiv}\bar{d} = c_1{\equiv}d_1{\wedge}...{\wedge}c_n{\equiv}d_n$ where $c_i, d_i{\in}cs|_p$.

The consistency of the construction is ensured if $\equiv$ and p[2] are in a syntactic form of executable specifications, thus that meaningful models exists if they exists for the concrete specification.

For the proof of the above obligations the user may use the specification C and the axioms added in steps

1,..,2 for proof obligation 2.
So the additional axiom of step 4. describes a proper submodel of C.

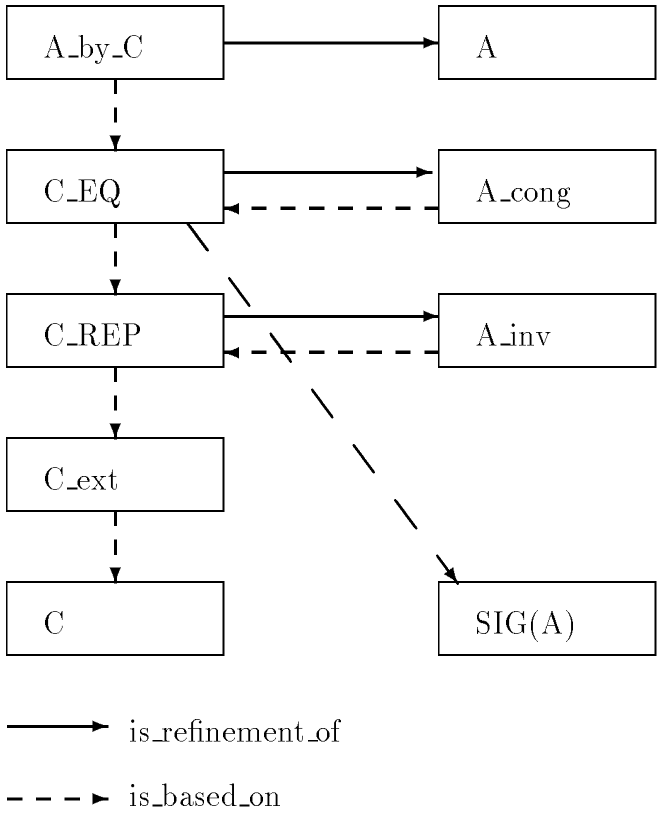
3,..,7 for proof obligation 3.
So the quotient of step 8. is well defined.

1,..,8 and the new generator of step 9. for proof obligation 1.

This provides a natural structure for the specifications.

2.3 Development Graph

The logical structure of the involved axioms and proof obligations may be depicted in the following KorSo development graph which are described in [BW93]. This complex structure is not necessary in the case of equational logic, but helps in structuring the proof and it is expected to carry over to more expressive logics (where it will be necessary).



[2]For p only consistency is required, since it will never be evaluated.

Where the contents of the specifications are:

C_ext contains the appropriate concrete specification C which may be an extension of a specification C.

C_REP contains C_ext plus axioms of step 1. - 2.

C_EQ contains C_REP plus axioms of step 3. - 7.

A_by_C contains C_EQ plus axioms of step 8. and 9.

A_inv enriches C_REP and contains proof obligation 2.

A_cong enriches C_EQ and contains proof obligation 3.

A contains A, for proof obligation 1.

SIG(A) contains the signature Σ_A.

2.4 Executability of the Construction

Checking executability with some necessary conditions is totally syntactic, and thus yields no proof obligations.

For executability of the whole construction the corresponding functions in C_ext have to be executable. In addition the definition of the equality $\equiv$ of C_EQ has to be executable, since it is computed every time to abstract sorts are tested on equality. This may be performed by an instantiation mechanism as in the functional programming language Haskell [HJW92],(Gofer [Jon93]), which allows to instantiate the member function of type classes.

3 Example Set by Sequences

In this section the general method of Section 2 is instantiated to the example of developing sets into sequences.

3.1 Specifications

Refining sets by sequences is a good example since it is small and contains all relevant details as subalgebras and quotients. The chosen representation of sets by sequences is the following:

- Every sequence without duplicates represents a set.
- Two sequences represent the same set if they contain the same elements.

This implementation decisions have to be filled in the generated schemes by the user of the method. For the notation of signatures and axioms the syntax of the specification language SPECTRUM [BFG+93a] is used, since it gives possibilities to describe axioms in a convenient way (even if it has no equational semantic). For example the definedness of a function $\delta(\mathsf{x}) \Rightarrow \delta(\mathsf{f}(\mathsf{x}))$ is abbreviated by f total; . The strictness of some functions is a requirement for specific functional implementation (ML).

3.1.1 Set

Set, the abstract sort (*as*) is specified polymorphically over the type class EQ. Axioms {set4} and {set5} require some sets to be equal. Sets are generated by *empty* and *add*.

```
SET = {
  sort Set α;
  Set::(EQ)EQ;
  empty: α::EQ ⇒ Set α;
  add: α::EQ ⇒ α × Set α → Set α;
  has: α::EQ ⇒ α × Set α → Bool;
  has total;
  add strict total;
  −− SET Axioms:
  Set α generated by empty, add;
  axioms α::EQ ⇒ ∀ x,y:α, s:Set α in
    {set1}   ¬(has(x,empty));
    {set2}   has(x,add(x,s));
    {set3}   ¬(x==y) ⇒ has(x,add(y,s)) = has(x,s);
    {set4}   add(x,add(x,s)) = add(x,s);
    {set5}   add(x,add(y,s)) = add(y,add(x,s));
  endaxioms;
}
```

3.1.2 Sequence

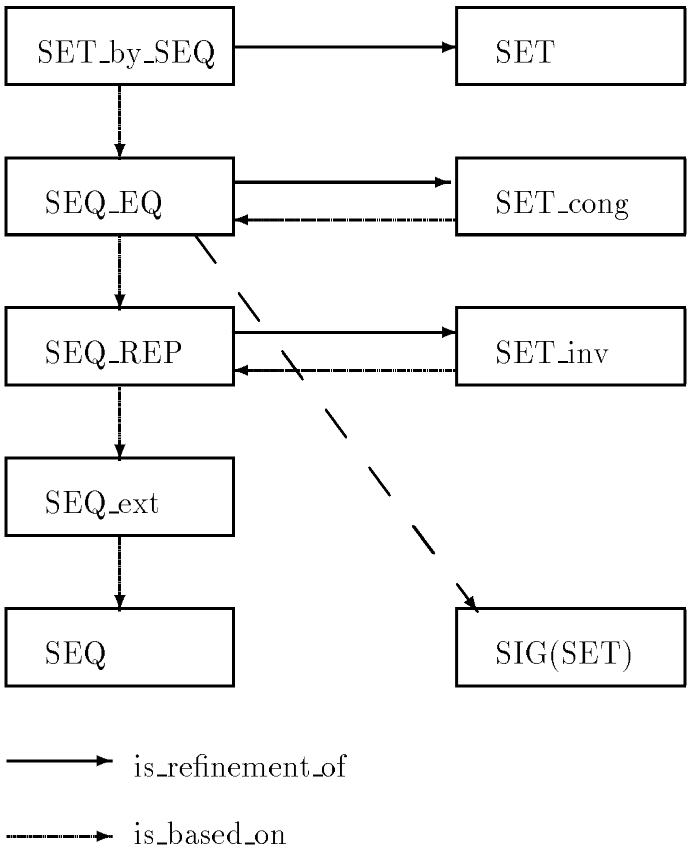
The concrete sort (*cs*) is sequence. The data-construct defines it together with constructor and selector functions and some further axioms which make the definition equivalent to the datatype declaration of ML.

```
SEQ = { strict;total;
        data Seq α = eseq
                   | cons(!first:α,!rest: Seq α);
        Seq::(EQ)EQ;

        isin: α::EQ ⇒ α × Seq α → Bool;
```

```
        subset: α::EQ ⇒ Seq α × Seq α → Bool;

        axioms α::EQ ⇒ ∀ x,y:α, p,q:Seq α in
        {isin1}   isin(x,eseq) = false;
        {isin2}   isin(x,cons(y,q)) = if x==y
                                       then true
                                       else isin(x,q)
                                       endif;
        {subs1}   subset(eseq,q) = true;
        {subs2}   subset(cons(x,p),q) = if isin(x,q)
                                          then subset(p,q)
                                          else false
                                          endif;
        endaxioms;
}
```

3.2 Generated Schemes

After the selection of SET and SEQ the following schemes are generated.



3.2.1 Extension of Sequences for Sets

In this specification the corresponding functions are specified. For syntactic reasons the $*$ is replaced by a suffix _x.

```
SEQ_ext = { enriches SEQ;

-- Step 0 (scheme): Define homomorphic images for every function of SET
          empty_x: α::EQ ⇒ Seq α;
          add_x: α::EQ ⇒ α × Seq α → Seq α;          add_x strict total;
          has_x: α::EQ ⇒ α × Seq α → Bool;           has_x total;

-- Step 0 (user): Specify the functions
          axioms α::EQ ⇒ ∀ x:α, p,q:Seq α in
          {construct_1}     empty_x = eseq;
          {construct_2}     add_x(x,q) = if isin(x,q)
                                            then q
                                            else cons(x,q)
                                         endif;
          {function_1}      has_x(x,q) = isin(x,q);
          endaxioms;
}
```

3.2.2 Restriction of Sequences for Sets

The representation predicate p is modelled by a function is_Set: Seq $\alpha \rightarrow$ Bool;

```
SEQ_REP = { enriches SEQ_ext;
-- Step 1 (scheme): Add the representation predicate:
      is_Set: α::EQ ⇒ Seq α → Bool; is_Set strict total;

-- Step 2 (user): Specify the representation predicate:
      axioms α::EQ ⇒ ∀ x:α, q:Seq α in
      {rep1}  is_Set(eseq);
      {rep2}  (is_Set(cons(x,q))):Bool = (isin(x,q)=false ∧ is_Set(q));
      endaxioms;
}
```

3.2.3 Homomorphism and Equality

The homomorphism is modelled by a partial function abs:Seq $\alpha \rightarrow$ Set α; whereas the congruence relation for the equality on sets is defined as a partial function eq_Set: Seq

$\alpha \times$ Seq $\alpha \to$ Bool; Both have to be defined only for elements with fulfil the restriction predicate is_Set.

In step 5 the arguments $\bar{c}$ and $\tilde{c}$ of the homomorphism are: $\bar{c} =$ x:α, q:Seq α and $\tilde{c} =$ q:Seq α.

```
SEQ_EQ = { enriches SEQ_REP + SIG_SET;
-- Step 3 (scheme): Define the (partial) homomorphism
      abs: Seq α → Set α; abs strict;

-- Step 4 (scheme): Give the source of the partial homomorphism abs
      axioms α::EQ ⇒ ∀ q:Seq α in
      {partial}      δ(abs(q))⇔is_Set(q);
      endaxioms;

-- Step 5 (scheme): Define the homomorphism:
      axioms α::EQ ⇒ ∀ x:α, q:Seq α in
      {hom1}             empty = abs(empty_x);
      {hom2} is_Set(q) ⇒ add(x,abs(q)) = abs(add_x(x,q));
      {hom3} is_Set(q) ⇒ has(x,abs(q)) = has_x(x,q);
      endaxioms;

-- Step 6 (scheme): Add a congruence predicate
      eq_Set: α::EQ ⇒ Seq α × Seq α → Bool; eq_Set strict;

-- Step 7 (user): Specify the congruence predicate
      axioms α::EQ ⇒ ∀ p,q:Seq α in
             -- in this case the following premise is not needed:
      {eq}    -- is_Set(p) ∧ is_Set(q) ⇒
                  eq_Set(p,q)=(subset(p,q) ∧ subset(q,p));
      endaxioms;
}
```

3.2.4 Generate Sets by Sequences

This specification is essential: it instantiates the equality to the congruence and adds the new generation principle.

```
SET_by_SEQ = { enriches SEQ_EQ;
-- Step 8 (scheme): Add the axiom to instantiate equality to the congruence
      axioms α::EQ ⇒ ∀ p,q:Seq α in
      {instant}      is_Set(p) ∧ is_Set(q) ⇒
                                  (abs(p)=abs(q)):Bool = (eq_Set(p,q));
      endaxioms;
```

```
−− Step 9 (scheme): Add the generation principle for sets
      Set α generated by abs;
}
```

3.2.5 Invariance of the Corresponding Functions

This specification contains all proof obligations for the invariance of the corresponding functions.

```
SET_inv = { enriches SEQ_REP;
−− invariance of the corresponding functions (proof obligation 2)
          axioms α::EQ ⇒ ∀ x:α, q:Seq α in
          {invar1}            is_Set(empty_x);
          {invar2}            is_Set(q) ⇒ is_Set(add_x(x,q));
          endaxioms;
}
```

3.2.6 Congruence of the Equality for the Corresponding Functions

This specification contains all proof obligations for the congruence of the equality for the corresponding functions.

```
SET_cong = { enriches SEQ_EQ;
−− the axioms for a congruence (proof obligation 3)
      axioms α::EQ ⇒ ∀ x:α, p,q,r:Seq α in
      {refl}         is_Set(p)            ⇒ eq_Set(p,p);
      {sym}          is_Set(p) ∧ is_Set(q) ⇒ eq_Set(p,q) = eq_Set(q,p);
      {trans}        is_Set(p) ∧ is_Set(q) ∧ is_Set(r) ∧
                            eq_Set(p,q) ∧ eq_Set(q,r) ⇒ eq_Set(p,r);
      {cong_add_x}   is_Set(p) ∧ is_Set(q) ∧
                            eq_Set(p,q) ⇒ eq_Set(add_x(x,p),add_x(x,q));
      {cong_has_x}   is_Set(p) ∧ is_Set(q) ∧
                            eq_Set(p,q) ⇒ has_x(x,p) = has_x(x,q);
      endaxioms;
}
```

3.3 Theories of the Specifications

The SPECTRUM specifications are translated automatically to the Isabelle HOLCF [Reg94] logic for SPECTRUM, called SPECHOLCF which supports partial functions and higher order axioms. This allows not only to do equational reasoning, but also to deduce the generated by axioms, which are not first order. The other proofs are first order proofs. This is done since the future work will be to carry over the change of data structures into an higher order logic as HOLCF.

The following theories (in Isabelle syntax) express the logical meaning of the specifications of the previous section. They are the bases for the reasoning.

3.3.1 Sets

The statement Set generated by empty, add; is according to the semantics of SPECTRUM [BFG+93b] translated to the axioms Set_Gen and Set_Exh. The annotations strict; and total; are translated to the axioms SET_Strict and SET_Total. SET_Def introduces the definedness of the function names.

```
SET = SPECHOLCF +


types

"Set_"            1


arities

"Set_"            :: (pcpo)pcpo

"Set_"            :: (eq)eq


consts

"empty"           :: "'a::eq Set_"

"add"             :: "('a::eq * 'a::eq Set_) -> 'a::eq Set_"

"has"             :: "('a::eq * 'a::eq Set_) -> tr"


rules

Set_Gen           "[|adm(P1);P1(UU);P1(empty); \
\                             !!y21 y22.[|add[y21#y22] ~= UU;P1(y22)|] \
\                                 ==> P1(add[y21#y22])|] ==> \
\                  (! x1.P1(x1))"

Set_Exh           "x = UU | x = empty | \
\                  (? y1 y2. y1 ~= UU & y2 ~= UU & x = add[y1#y2])"
```

```
set1                "x ~= UU ==> \
\                    not(has[x#empty] = TT)"

set2                "[|x ~= UU;s ~= UU|] ==> \
\                    has[x#add[x#s]] = TT"

set3                "[|x ~= UU;y ~= UU;s ~= UU|] ==> \
\                    not((x === y) = TT) --> has[x#add[y#s]] = has[x#s]"

set4                "[|x ~= UU;s ~= UU|] ==> \
\                    add[x#add[x#s]] = add[x#s]"

set5                "[|x ~= UU;y ~= UU;s ~= UU|] ==> \
\                    add[x#add[y#s]] = add[y#add[x#s]]"

SET_Def             "empty ~= UU & add ~= UU & has ~= UU"

SET_Strict          "(! x1 x2. (x1 = UU) | (x2 = UU) --> \
\                    (add[x1#x2] = UU))"

SET_Total           "(! x1 x2. (x1 ~= UU) & (x2 ~= UU) --> \
\                    (has[x1#x2] ~= UU)) & \
\                    (! x1 x2. (x1 ~= UU) & (x2 ~= UU) --> (add[x1#x2] ~= UU))"


end
```

3.3.2 Sequences

The data construct is translated into several functions and axioms. For details see [BFG+93a].

```
SEQ = SPECHOLCF +


types

"Seq"               1


arities

"Seq"               :: (pcpo)pcpo

"Seq"               :: (eq)eq
```

```
consts

"Seq_When"          :: "'w -> ('a -> 'a Seq -> 'w) -> 'a Seq -> 'w"

"eseq"              :: "'a Seq"

"is_eseq"           :: "'a Seq -> tr"

"cons"              :: "('a * 'a Seq) -> 'a Seq"

"is_cons"           :: "'a Seq -> tr"

"first"             :: "'a Seq -> 'a"

"rest"              :: "'a Seq -> 'a Seq"

"isin"              :: "('a::eq * 'a::eq Seq) -> tr"

"subset"            :: "('a::eq Seq * 'a::eq Seq) -> tr"


rules

Seq_Gen             "[|adm(P1);P1(UU);P1(eseq); \
\                               !!y21 y22.[|cons[y21#y22] ~= UU;P1(y22)|] \
\                                   ==> P1(cons[y21#y22])|] ==> \
\                    (! x1.P1(x1))"

Seq_Exh             "x = UU | x = eseq | \
\                    (? y1 y2. y1 ~= UU & y2 ~= UU & x = cons[y1#y2])"

Seq_When            "(Seq_When[f1][f2][UU] = UU) & \
\                    ((eseq ~= UU) --> Seq_When[f1][f2][eseq] = f1) & \
\                    ((cons[x1#x2] ~= UU) --> \
\                    Seq_When[f1][f2][cons[x1#x2]] = f2[x1][x2])"

Seq_Select          "first = Seq_When[UU][(LAM x1. (LAM x2. x1))] & \
\                    rest = Seq_When[UU][(LAM x1. (LAM x2. x2))]"

Seq_Constrdef       "(eseq~=UU) & \
\                    (! x1 x2. (cons[x1#x2]~=UU) = ((x1~=UU) & (x2~=UU)))"

Seq_Discrim         "is_eseq = Seq_When[TT][(LAM x1. (LAM x2. FF))] & \
```

```
\                    is_cons = Seq_When[FF][(LAM x1. (LAM x2. TT))]"

isin1               "x ~= UU ==> \
\                    isin[x#eseq] = FF"

isin2               "[|x ~= UU;y ~= UU;q ~= UU|] ==> \
\                    isin[x#cons[y#q]] = \
\                    If x === y then TT else isin[x#q] fi"

subs1               "q ~= UU ==> \
\                    subset[eseq#q] = TT"

subs2               "[|x ~= UU;p ~= UU;q ~= UU|] ==> \
\                    subset[cons[x#p]#q] = \
\                    If isin[x#q] then subset[p#q] else FF fi"

SEQ_Def             "Seq_When ~= UU & eseq ~= UU & is_eseq ~= UU & \
\                    cons ~= UU & is_cons ~= UU & first ~= UU & \
\                    rest ~= UU & isin ~= UU & subset ~= UU"

SEQ_Strict          "(! x1 x2. (x1=UU) | (x2=UU) --> (isin[x1#x2]=UU)) & \
\                    (! x1 x2. (x1=UU) | (x2=UU) --> (subset[x1#x2]=UU))"

SEQ_Total           "(! x1 x2. (x1~=UU) & (x2~=UU) --> (isin[x1#x2]~=UU)) & \
\                    (! x1 x2. (x1~=UU) & (x2~=UU) --> (subset[x1#x2]~=UU))"


end
```

3.3.3 Extension of Sequences for Sets

```
SEQ_ext = SEQ +


consts

"empty_x"           :: "'a::eq Seq"

"add_x"             :: "('a::eq * 'a::eq Seq) -> 'a::eq Seq"

"has_x"             :: "('a::eq * 'a::eq Seq) -> tr"


rules
```

```
construct_1         "empty_x = eseq"

construct_2         "[|x ~= UU;q ~= UU|] ==> \
\                    add_x[x#q] = If isin[x#q] then q else cons[x#q] fi"

function_1          "[|x ~= UU;q ~= UU|] ==> \
\                    has_x[x#q] = isin[x#q]"

SEQ_ext_Def         "empty_x ~= UU & add_x ~= UU & has_x ~= UU"

SEQ_ext_Strict      "(! x1 x2. (x1 = UU) | (x2 = UU) --> (add_x[x1#x2] = UU))"

SEQ_ext_Total       "(! x1 x2. (x1~=UU) & (x2~=UU) --> (add_x[x1#x2]~=UU)) &\
\                    (! x1 x2. (x1~=UU) & (x2~=UU) --> (has_x[x1#x2]~=UU))"


end
```

3.3.4 Restriction of Sequences for Sets

```
SEQ_REP = SEQ_ext +


consts

"is_Set"            :: "'a::eq Seq -> tr"


rules

rep1                "is_Set[eseq] = TT"

rep2                "[|x ~= UU;q ~= UU|] ==> \
\                    (is_Set[cons[x#q]] = TT) = \
\                    (isin[x#q] = FF & is_Set[q] = TT)"

SEQ_REP_Def         "is_Set ~= UU"

SEQ_REP_Strict      "(! x1. (x1 = UU) --> (is_Set[x1] = UU))"

SEQ_REP_Total       "(! x1. (x1 ~= UU) --> (is_Set[x1] ~= UU))"


end
```

3.3.5 Homomorphism and Equality

SIG_SET contains only the signature of SET and the strictness axiom of the function add.

```
SEQ_EQ = SEQ_REP + SIG_SET +


consts

"abs"              :: "'a Seq -> 'a Set_"

"eq_Set"           :: "('a::eq Seq * 'a::eq Seq) -> tr"


rules

partial            "q ~= UU ==> \
\                   ((abs[q]) ~= UU) <-> is_Set[q] = TT"

hom1               "empty = abs[empty_x]"

hom2               "[|q ~= UU;x ~= UU|] ==> \
\                   is_Set[q] = TT --> add[x#abs[q]] = abs[add_x[x#q]]"

hom3               "[|q ~= UU;x ~= UU|] ==> \
\                   is_Set[q] = TT --> has[x#abs[q]] = has_x[x#q]"

eq                 "[|p ~= UU;q ~= UU|] ==> \
\                   eq_Set[p#q] = (subset[p#q] parand subset[q#p])"

SEQ_EQ_Def         "abs ~= UU & eq_Set ~= UU"

SEQ_EQ_Strict      "(! x1. (x1 = UU) --> (abs[x1] = UU)) & \
\                   (! x1 x2. (x1 = UU) | (x2 = UU) --> (eq_Set[x1#x2] = UU))"


end
```

3.3.6 Generate Sets by Sequences

```
SET_by_SEQ = SEQ_EQ +


rules
```

```
instant              "[|p ~= UU;q ~= UU|] ==> \
\                     is_Set[p] = TT & is_Set[q] = TT --> \
\                     (abs[p] = abs[q]) = (eq_Set[p#q] = TT)"

Set_Gen              "[|adm(P1);P1(UU);!!y11.[|abs[y11] ~= UU|] \
\                                   ==> P1(abs[y11])|] ==> \
\                     (! x1.P1(x1))"

Set_Exh              "x = UU | (? y1. y1 ~= UU & x = abs[y1])"


end
```

3.3.7 Invariance of the Corresponding Functions

```
SET_inv = SEQ_REP +


rules

invar1               "is_Set[empty_x] = TT"

invar2               "[|q ~= UU;x ~= UU|] ==> \
\                     is_Set[q] = TT --> is_Set[add_x[x#q]] = TT"


end
```

3.3.8 Congruence of the Equality for the Corresponding Functions

```
SET_cong = SEQ_EQ +


rules

refl                 "p ~= UU ==> \
\                     is_Set[p] = TT --> eq_Set[p#p] = TT"

sym                  "[|p ~= UU;q ~= UU|] ==> \
\                     is_Set[p] = TT & is_Set[q] = TT --> \
\                     eq_Set[p#q] = eq_Set[q#p]"

trans                "[|p ~= UU;q ~= UU;r ~= UU|] ==> \
\                     is_Set[p] = TT & \
\                     is_Set[q] = TT & \
\                     is_Set[r] = TT & eq_Set[p#q] = TT & eq_Set[q#r] = TT\
```

```
\                     --> eq_Set[p#r] = TT"

cong_add_x         "[|p ~= UU;q ~= UU;x ~= UU|] ==> \
\                   is_Set[p] = TT & is_Set[q] = TT & eq_Set[p#q] = TT --> \
\                   eq_Set[add_x[x#p]#add_x[x#q]] = TT"

cong_has_x         "[|p ~= UU;q ~= UU;x ~= UU|] ==> \
\                   is_Set[p] = TT & is_Set[q] = TT & eq_Set[p#q] = TT --> \
\                   has_x[x#p] = has_x[x#q]"


end
```

3.4 Proof obligations

Every *is_refinement_of* relation of the development graph has to be proved. The following proof frames are generated by a program which structures the proof such that for every axiom in the refined theory (specification) one lemma is used to prove it. The proof of every lemma is in an extra file which is loaded with `use`. If the proof of one lemma fails the frame aborts. If all lemmas are proved the frame generates the theorem.

3.4.1 SET with SET_by_SEQ

To prove the relation SET_by_SEQ *is_refinement_of* SET the following frame is generated:

```
(*  I S A B E L L E - P R O O F - F R A M E                         *)
(*               f o r  t h e                                       *)
(*    R e f i n e m e n t r e l a t i o n                           *)
(*                   o f                                            *)
(*    SET.thy  b y  SET_by_SEQ.thy                                  *)
(* ****************************************************************** *)
(* Loading general lemmas for SPECHOLCF                             *)
use "SPECHOLCF.ML";
(* ****************************************************************** *)
(* Loading the base-theory SET_by_SEQ.thy                           *)
use_thy "SET_by_SEQ";
(* ****************************************************************** *)
(* There is one theory to prove:                                    *)
(* ++++++++ SET ++++++++                                            *)
(*                                                                  *)
(* Goals for SET                                                    *)
(*                                                                  *)
val prems = goal SET_by_SEQ.thy "[|adm(P1);P1(UU);P1(empty); \
\                              !!y21 y22.[|add[y21#y22] ~= UU;P1(y22)|] \
```

```
\                                  ==> P1(add[y21#y22])|] ==> \
\                    (! x1.P1(x1))";
use "SET_by_SET_by_SEQ:Set_Gen.prf";
val SET_by_SET_by_SEQ_Set_Gen = result();

val prems = goal SET_by_SEQ.thy "x = UU | x = empty | \
\                    (? y1 y2. y1 ~= UU & y2 ~= UU & x = add[y1#y2])";
use "SET_by_SET_by_SEQ:Set_Exh.prf";
val SET_by_SET_by_SEQ_Set_Exh = result();

val prems = goal SET_by_SEQ.thy "x ~= UU ==> \
\                    not(has[x#empty] = TT)";
use "SET_by_SET_by_SEQ:set1.prf";
val SET_by_SET_by_SEQ_set1 = result();

val prems = goal SET_by_SEQ.thy "[|x ~= UU;s ~= UU|] ==> \
\                    has[x#add[x#s]] = TT";
use "SET_by_SET_by_SEQ:set2.prf";
val SET_by_SET_by_SEQ_set2 = result();

val prems = goal SET_by_SEQ.thy "[|x ~= UU;y ~= UU;s ~= UU|] ==> \
\                    not((x === y) = TT) --> has[x#add[y#s]] = has[x#s]";
use "SET_by_SET_by_SEQ:set3.prf";
val SET_by_SET_by_SEQ_set3 = result();

val prems = goal SET_by_SEQ.thy "[|x ~= UU;s ~= UU|] ==> \
\                    add[x#add[x#s]] = add[x#s]";
use "SET_by_SET_by_SEQ:set4.prf";
val SET_by_SET_by_SEQ_set4 = result();

val prems = goal SET_by_SEQ.thy "[|x ~= UU;y ~= UU;s ~= UU|] ==> \
\                    add[x#add[y#s]] = add[y#add[x#s]]";
use "SET_by_SET_by_SEQ:set5.prf";
val SET_by_SET_by_SEQ_set5 = result();

val prems = goal SET_by_SEQ.thy "empty ~= UU & add ~= UU & has ~= UU";
use "SET_by_SET_by_SEQ:SET_Def.prf";
val SET_by_SET_by_SEQ_SET_Def = result();

val prems = goal SET_by_SEQ.thy "(! x1 x2. (x1 = UU) | (x2 = UU) --> \
\                    (add[x1#x2] = UU))";
use "SET_by_SET_by_SEQ:SET_Strict.prf";
val SET_by_SET_by_SEQ_SET_Strict = result();

val prems = goal SET_by_SEQ.thy "(! x1 x2. (x1 ~= UU) & (x2 ~= UU) --> \
```

```
\                    (has[x1#x2] ~= UU)) & \
\                    (! x1 x2. (x1 ~= UU) & (x2 ~= UU) --> (add[x1#x2] ~= UU))";
use "SET_by_SET_by_SEQ:SET_Total.prf";
val SET_by_SET_by_SEQ_SET_Total = result();

(* ****************************************************************** *)
(* End of proof                                                        *)
output(open_out("SET_by_SET_by_SEQ.thm"),"proved");
quit();
```

3.4.2 Set's Congruence

To prove the relation SET_cong *is_refinement_of* SEQ_EQ the following frame is generated:

```
(*   I S A B E L L E - P R O O F - F R A M E                          *)
(*                for  t h e                                           *)
(*     R e f i n e m e n t r e l a t i o n                             *)
(*                    o f                                              *)
(*     SET_cong.thy  b y  SEQ_EQ.thy                                   *)
(* ****************************************************************** *)
(* Loading general lemmas for SPECHOLCF                                *)
use "SPECHOLCF.ML";
(* ****************************************************************** *)
(* Loading the base-theory SEQ_EQ.thy                                  *)
use_thy "SEQ_EQ";
(* ****************************************************************** *)
(* There is one theory to prove:                                       *)
(* ++++++++ SET_cong ++++++++                                          *)
(*                                                                     *)
(* Goals for SET_cong                                                  *)
(*                                                                     *)
val prems = goal SEQ_EQ.thy "p ~= UU ==> \
\                    is_Set[p] = TT --> eq_Set[p#p] = TT";
use "SET_cong_by_SEQ_EQ:refl.prf";
val SET_cong_by_SEQ_EQ_refl = result();

val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU|] ==> \
\                    is_Set[p] = TT & is_Set[q] = TT --> \
\                    eq_Set[p#q] = eq_Set[q#p]";
use "SET_cong_by_SEQ_EQ:sym.prf";
val SET_cong_by_SEQ_EQ_sym = result();

val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU;r ~= UU|] ==> \
\                    is_Set[p] = TT & \
\                    is_Set[q] = TT & \
```

```
\                    is_Set[r] = TT & eq_Set[p#q] = TT & eq_Set[q#r] = TT\
\                     --> eq_Set[p#r] = TT";
use "SET_cong_by_SEQ_EQ:trans.prf";
val SET_cong_by_SEQ_EQ_trans = result();

val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU;x ~= UU|] ==> \
\                    is_Set[p] = TT & is_Set[q] = TT & eq_Set[p#q] = TT --> \
\                    eq_Set[add_x[x#p]#add_x[x#q]] = TT";
use "SET_cong_by_SEQ_EQ:cong_add_x.prf";
val SET_cong_by_SEQ_EQ_cong_add_x = result();

val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU;x ~= UU|] ==> \
\                    is_Set[p] = TT & is_Set[q] = TT & eq_Set[p#q] = TT --> \
\                    has_x[x#p] = has_x[x#q]";
use "SET_cong_by_SEQ_EQ:cong_has_x.prf";
val SET_cong_by_SEQ_EQ_cong_has_x = result();

(* ****************************************************************** *)
(* End of proof                                                       *)
output(open_out("SET_cong_by_SEQ_EQ.thm"),"proved");
quit();
```

3.4.3 Set's Invariance

To prove the relation SET_inv *is_refinement_of* SEQ_REP the following frame is generated:

```
(*   I S A B E L L E - P R O O F - F R A M E                          *)
(*                 f o r   t h e                                      *)
(*     R e f i n e m e n t r e l a t i o n                            *)
(*                     o f                                            *)
(*     SET_inv.thy  b y  SEQ_REP.thy                                  *)
(* ****************************************************************** *)
(* Loading general lemmas for SPECHOLCF                                *)
use "SPECHOLCF.ML";
(* ****************************************************************** *)
(* Loading the base-theory SEQ_REP.thy                                *)
use_thy "SEQ_REP";
(* ****************************************************************** *)
(* There is one theory to prove:                                      *)
(* ++++++++ SET_inv ++++++++                                          *)
(*                                                                    *)
(* Goals for SET_inv                                                  *)
(*                                                                    *)
val prems = goal SEQ_REP.thy "is_Set[empty_x] = TT";
use "SET_inv_by_SEQ_REP:invar1.prf";
```

```
val SET_inv_by_SEQ_REP_invar1 = result();

val prems = goal SEQ_REP.thy "[|q ~= UU;x ~= UU|] ==> \
\                    is_Set[q] = TT --> is_Set[add_x[x#q]] = TT";
use "SET_inv_by_SEQ_REP:invar2.prf";
val SET_inv_by_SEQ_REP_invar2 = result();

(* ************************************************************* *)
(* End of proof                                                  *)
output(open_out("SET_inv_by_SEQ_REP.thm"),"proved");
quit();
```

4 Proof of the Example

This section contains all proofs for the change of data structures in the example of sets and sequences. The proofs were performed with the Isabelle proof system [Pau94], which gives nice support for proof management. The proof took about two weeks, where at least half of the effort was to prove general lemmas about SPECTRUM's logic and sequences. These lemmas and the developed proof method will simplify future proofs. There is still room for further improvement by writing special tactics for definedness reasoning or automatic expansion and reduction of homomorphisms. The aim of this section is to documentate the proofs and to give a method which helps to carry through such proofs.

4.1 Proof Method and Management

The proof method supports fulfilling the proof obligations arising from the change of data structures presented in the previous sections. Proof management provides tool support for developing theories and proving lemmas in a structured way.

Every specification in the graph of page 8 corresponds to one Isabelle theory. Specifications enrich other specifications and so do the corresponding theories. Isabelle loads with use_thy a theory with all enriched subtheories. So it takes only one command to load a theory into Isabelle.

    use_thy "SET_by_SEQ";

The correctness of the change of data structures requires three refinement proofs. The goal of each refinement is a theory, corresponding to the refined specification. The bases for these proofs is the theory of the refining specification, which is more concrete than the refined specifications. In the graphical notation of KORSO the refinement arrow points from the base to the goal (like the implication arrow).

A theory has a set of rules and all rules of the goal theory have to be proved in the refinement proof. This provides a good structure for refinement proofs. A simple tool generated the proof frames of Section 3.4. For every rule in the goal theory one lemma is generated in the frame. The developer puts the proof of the lemma in a file which is loaded by the frame with use. The frame checks with result() whether the proof is complete.

```
(* generate the goal for the rule set4: *)
     val prems = goal SET_by_SEQ.thy "[|x ~= UU;s ~= UU|] ==> \
     \                 add[x#add[x#s]] = add[x#s]";
(* load the file with the proof of the rule *)
     use "SET_by_SET_by_SEQ:set4.prf";
(* check whether the goal has been proved *)
     val SET_by_SET_by_SEQ_set4 = result();
```

This is done for every rule of the goal theory. If all lemmas are proved the frame generates the fact that the refinement is proved. In other cases it stops for completing the proof interactively.

```
(* End of proof *)
     output(open_out("SET_by_SET_by_SEQ.thm"),"proved");
```

Proving one lemma may require the application of many axioms. Some axioms (induction, case-split, ..) replace one goal by a number of different subgoals. Each of them has to be proved. These subgoals are managed by the Isabelle proof system. A typical proof state for an induction proof on sequences is:

```
> # Level 2
   (* main goal *)
subset[p#q] = TT --> subset[p#cons[x#q]] = TT
   (* subgoals *)
      (* admissibility *)
 1. [| p ~= UU; q ~= UU; x ~= UU |] ==>
   adm(%x1. subset[x1#q] = TT --> subset[x1#cons[x#q]] = TT)
      (* undefined case *)
 2. [| p ~= UU; q ~= UU; x ~= UU |] ==>
   subset[UU#q] = TT --> subset[UU#cons[x#q]] = TT
      (* case eseq *)
 3. [| p ~= UU; q ~= UU; x ~= UU |] ==>
   subset[eseq#q] = TT --> subset[eseq#cons[x#q]] = TT
      (* case cons: induction step *)
 4. !!y21 y22.
      [| p ~= UU; q ~= UU; x ~= UU; cons[y21#y22] ~= UU;
        subset[y22#q] = TT --> subset[y22#cons[x#q]] = TT |] ==>
      subset[cons[y21#y22]#q] = TT --> subset[cons[y21#y22]#cons[x#q]] = TT
```

The method for almost all occurring proofs are induction and case split. Case split usually generates three subgoals, since we have three boolean values (UU,TT,FF). For induction proofs the admissibility of the predicate is the first thing to prove. This is quite schematic on flat domains, since all predicates on flat domain are admissible.

The *invariance proofs* (see Section 4.5) are performed along the definitions of the corresponding functions (see SEQ_ext at page 15) with the simplifier. empty_x has a direct translation, so the invariance is proved in one step. add_x is defined with a case distinction on isin, so the proof follows this syntactic form and does a case split on isin to.

The *congruence proofs* (see Section 4.4) apply the definitions of the equality eq_Set (see SEQ_REP at page 16). The equivalence relation properties are shown by exploiting the knowledge about sequences. These general lemmas are stored in SEQ.ML (see page 29). For the substitutivity the case split on isin is done since the definition of add_x is applied. Additionally some forward reasoning is done exploiting the definedness of the premises (e.g. is_Set(q) ==> is_Set(add_x(x,q))).

The *set proofs* are the axioms of the specification SET (see Section 4.3) with variables ranging over Set α. These axioms may be proved by induction on sets exploiting the fact that abs is surjective on sets. This general lemma for SET_by_SEQ is proved at first (Set_surjectiv at page 35). With this general lemma all proofs on sets may use induction on sequences instead of induction on sets, even the proof of the induction rule on sets may be performed by induction on Sequences (see Section 4.3.1). The rest of the axioms require some case distinction on isin as in the other parts of the proof.

4.2 General Lemmas

For structuring the proof some general lemmas were proved. Many of them are reused in several places of the other proofs. The lemmas are divided into the theories they belong to. Isabelle automatically loads the lemmas to the corresponding theories and replays the proofs.

4.2.1 Lemmas for SPECHOLCF

General lemmas for SPECHOLCF simplify the proof technique somewhere in the proofs.

```
open SPECHOLCF;

val weq_refl = refl RSN (3,weq RS conjunct1 RS mp);
val case_split = trE;

val SPECHOLCF_ss = ccc1_ss addsimps (one_when @ dist_less_one @ dist_eq_one @
dist_less_tr @ dist_eq_tr @ tr_when @ andalso_thms @ orelse_thms @
ifte_thms @ [neg_def,parand_def, paror_def,biimpl_def, strict_weq1,
strict_weq2, total_weq, weq_refl, defALL_def, defEX_def]);

val not_UU_weq = prove_goal SPECHOLCF.thy "[|x~=UU;y~=UU;x===y~=TT|] ==> \
\  x===y=FF"
```

```
 (fn prems =>
[
(cut_facts_tac prems 1),
(forw_inst_tac [("x","x"),("y","y")] total_weq 1),
(atac 1),
(res_inst_tac [("p","x===y")] case_split 1),
(REPEAT (fast_tac HOL_cs 1))
]);

val if_total = prove_goal SPECHOLCF.thy
"[|b~=UU;x~=UU;y~=UU|]==>If b then x else y fi ~=UU"
 (fn prems =>
[
(cut_facts_tac prems 1),
(res_inst_tac [("p","b")] case_split 1),
(rtac notE 1),
(atac 2),
(REPEAT (asm_simp_tac SPECHOLCF_ss 1))
]);

val if_eq = prove_goal SPECHOLCF.thy "b~=UU==>If b then x else x fi =x"
 (fn prems =>
[
(cut_facts_tac prems 1),
(res_inst_tac [("p","b")] case_split 1),
(fast_tac HOL_cs 1),
(REPEAT (asm_simp_tac SPECHOLCF_ss 1))
]);

val if_eq_TT = prove_goal SPECHOLCF.thy "If b then c else FF fi = TT ==> \
\ b=TT & c=TT"
 (fn prems =>
[
(res_inst_tac [("p","b")] case_split 1),
(cut_facts_tac prems 1),
(asm_full_simp_tac SPECHOLCF_ss 1),
(cut_facts_tac prems 1),
(asm_full_simp_tac SPECHOLCF_ss 1),
(cut_facts_tac prems 1),
(asm_full_simp_tac SPECHOLCF_ss 1)
]);

val if_exp = prove_goal SPECHOLCF.thy "If b then TT else FF fi = b"
 (fn prems =>
[
```

```
(res_inst_tac [("p","b")] case_split 1),
(REPEAT(asm_full_simp_tac SPECHOLCF_ss 1))
]);

val eq_disj = prove_goal HOL.thy "x=y | x~=y"
 (fn prems => [ (fast_tac HOL_cs 1) ]);

val cont_inject = prove_goal SPECHOLCF.thy "[|f[x]=FF;f[y]=TT|]==>x~=y"
 (fn prems =>
[
(res_inst_tac [("x1","x"),("y1","y")] (eq_disj RS disjE) 1),
(cut_facts_tac prems 1),
(asm_full_simp_tac SPECHOLCF_ss 1),
(atac 1)
]);

val not_eq_sym = prove_goal HOL.thy "x~=y ==> y~=x"
 (fn prems =>
[
(cut_facts_tac prems 1),
(fast_tac HOL_cs 1)
]);

val weq_sym = prove_goal SPECHOLCF.thy "[|x~=UU; y~=UU |] ==> x===y = (y===x)"
 (fn prems =>
[
(cut_facts_tac prems 1),
(res_inst_tac [("x1","x"),("y1","y")] (eq_disj RS disjE) 1),
(asm_full_simp_tac SPECHOLCF_ss 1),
(forw_inst_tac [("x","x"),("y","y")] not_eq_sym 1),
(forw_inst_tac [("x2","x"),("y2","y")] (weq RS conjunct2 RS mp) 1),
(forw_inst_tac [("x2","y"),("y2","x")] (weq RS conjunct2 RS mp) 3),
(REPEAT (atac 1)),
(asm_full_simp_tac SPECHOLCF_ss 1)
]);

val parand_sym = prove_goal SPECHOLCF.thy "(x parand y) = (y parand x)"
 (fn prems =>
[
(res_inst_tac [("p","x")] case_split 1),
(res_inst_tac [("p","y")] case_split 3),
(res_inst_tac [("p","y")] case_split 2),
(res_inst_tac [("p","y")] case_split 1),
(REPEAT (asm_simp_tac SPECHOLCF_ss 1))
]);
```

```
val parand_TT2 = prove_goal SPECHOLCF.thy "x=TT & y=TT --> (x parand y) = TT"
 (fn prems =>
[
(res_inst_tac [("p","x")] case_split 1),
(res_inst_tac [("p","y")] case_split 3),
(res_inst_tac [("p","y")] case_split 2),
(res_inst_tac [("p","y")] case_split 1),
(REPEAT (asm_simp_tac SPECHOLCF_ss 1))
]);


val parand_TT = prove_goal SPECHOLCF.thy "(x parand y) = TT --> x=TT & y=TT"
 (fn prems =>
[
(res_inst_tac [("p","x")] case_split 1),
(res_inst_tac [("p","y")] case_split 3),
(res_inst_tac [("p","y")] case_split 2),
(res_inst_tac [("p","y")] case_split 1),
(REPEAT (asm_simp_tac SPECHOLCF_ss 1))
]);

val UU_or_not_UU = prove_goal SPECHOLCF.thy "x=UU|x~=UU"
 (fn prems => [ (fast_tac HOL_cs 1) ]);

val TT_imp_not_FF = prove_goal SPECHOLCF.thy "x=TT==>x~=FF"
 (fn prems =>
[
(cut_facts_tac prems 1),
(asm_simp_tac SPECHOLCF_ss 1)
]);

val FF_imp_not_TT = prove_goal SPECHOLCF.thy "x=FF==>x~=TT"
 (fn prems =>
[
(cut_facts_tac prems 1),
(asm_simp_tac SPECHOLCF_ss 1)
]);

val not_eq_UU_lemma = prove_goal SPECHOLCF.thy "[|x~=UU=(s=t);s~=t|]==>x=UU"
 (fn prems =>
[
(cut_facts_tac prems 1),
(cut_facts_tac [hd(tl(tl(dist_eq_tr)))] 1),
(fast_tac HOL_cs 1)
```

```
]);

val not_UU_trans = prove_goal SPECHOLCF.thy "[|x~=UU;x=y|]==>y~=UU"
 (fn prems =>
[
(cut_facts_tac prems 1),
(fast_tac HOL_cs 1)
]);

fun subgoal_tac_ss ss = assume_tac ORELSE'
resolve_tac (prems_of_ss ss) ORELSE'
asm_simp_tac ss ORELSE'
fast_tac HOL_cs;
val S_ss = SPECHOLCF_ss setsubgoaler subgoal_tac_ss;
```

4.2.2 Lemmas for SEQ

This lemmas talk about properties and functions of sequences.

```
val prems = goal SEQ.thy "[|x~=UU;q~=UU|] ==> cons[x#q]~=UU";
by (cut_facts_tac prems 1);
by (asm_simp_tac (HOLCF_ss addsimps [SEQ.Seq_Constrdef RS conjunct2]) 1);
val cons_Total1 = result();

val prems = goal SEQ.thy "cons[x#q]~=UU ==> x~=UU & q~=UU";
by (cut_facts_tac prems 1);
br subst 1;
br (SEQ.Seq_Constrdef RS conjunct2 RS spec RS spec) 1;
ba 1;
val cons_strict = result();

val c1 = conjunct1;
val c2 = conjunct2;
val isin_total = conjI RS (SEQ.SEQ_Total RS c1 RS spec RS spec RS mp);
val subset_total = conjI RS (SEQ.SEQ_Total RS c2 RS spec RS spec RS mp);
val def_eseq = SEQ.SEQ_Def RS c2 RS c1;
val subset_strict1 = disjI1 RS (SEQ.SEQ_Strict RS c2 RS spec RS spec RS mp);
val subset_strict2 = disjI2 RS (SEQ.SEQ_Strict RS c2 RS spec RS spec RS mp);

goal SEQ.thy "subset[x#UU] = UU";
br subset_strict2 1;
br refl 1;
val subset_UU2 = result();

goal SEQ.thy "subset[eseq#UU] = UU";
br subset_UU2 1;
```

```
val subset_eseq_UU = result();

goal SEQ.thy "subset[UU#x] = UU";
br subset_strict1 1;
br refl 1;
val subset_UU1 = result();

val SEQ_Prg = [SEQ.isin1,SEQ.isin2,SEQ.subs1,SEQ.subs2,subset_UU2,subset_UU1];
val SEQ_Data = [SEQ.SEQ_Total,SEQ.SEQ_Strict,SEQ.Seq_When,SEQ.Seq_Select,
 SEQ.Seq_Constrdef,SEQ.Seq_Discrim,SEQ.SEQ_Def];

val prems = goal SEQ.thy
"[| p~=UU; q~=UU; x~=UU |] ==> subset[p#q]=TT --> subset[p#cons[x#q]]=TT";
by (cut_facts_tac prems 1);
(* by induction on Seq generated by eseq, cons *)
by (res_inst_tac [("x","p")] (SEQ.Seq_Gen RS spec) 1);
by (rtac adm_flat 1);
by (rtac flat_lemma 1);
   (* case UU *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case eseq *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case cons *)
by (rtac impI 1);
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
by (dtac if_eq_TT 1);
by (asm_full_simp_tac (S_ss addsimps ([if_eq]@SEQ_Prg@SEQ_Data)) 1);
val SEQ_lem1=result();

val prems = goal SEQ.thy "q ~= UU ==> subset[q#q] = TT";
(* convert to --> *)
by (rtac mp 1);
by (cut_facts_tac prems 2);
by (atac 2);
(* by induction on Seq generated by eseq, cons *)
by (rtac (SEQ.Seq_Gen RS spec) 1);
by (rtac adm_flat 1);
by (rtac flat_lemma 1);
   (* case UU *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case eseq *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case cons: complete with lemma 1*)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data@[SEQ_lem1])) 1);
val SEQ_lem2=result();
```

```
val prems = goal SEQ.thy
"[|x~=UU;p~=UU;q~=UU|] ==> isin[x#p]=FF&subset[q#p]=TT --> isin[x#q]=FF";
(* convert to --> *)
br mp 1;
by (cut_facts_tac prems 2);
be conjI 2;
be conjI 2;
ba 2;
(* by induction on Seq generated by eseq, cons *)
by (res_inst_tac [("x","q")] (SEQ.Seq_Gen RS spec) 1);
br adm_flat 1;
br flat_lemma 1;
   (* case UU *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case eseq *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case cons *)
      (* definedness reasoning *)
br impI 1;
by (forward_tac [c1] 1);
bd c2 1;
by (forward_tac [c1] 1);
bd c2 1;
by (forward_tac [cons_strict RS c1] 1);
by (forward_tac [cons_strict RS c2] 1);
br impI 1;
by (forward_tac [c1] 1);
bd c2 1;
        (* subgoals x===y21=FF and isin[x#y22]=FF*)
by (subgoal_tac "x===y21=FF & isin[x#y22]=FF" 1);
        (* solve cons-case *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
        (* solve subgoals *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
bd if_eq_TT 1;
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
by (res_inst_tac [("x2","x"),("y2","y21"),("f3","LAM x.isin[x#p]")]
        (cont_inject RSN (3,(weq RS c2 RS mp))) 1);
by (REPEAT (asm_full_simp_tac S_ss 1));
val SEQ_lem3 = result();

val prems = goal SEQ.thy
"[|x~=UU;p~=UU;q~=UU|] ==> isin[x#p]=TT&subset[p#q]=TT --> isin[x#q]=TT";
by (cut_facts_tac prems 1);
```

```
(* by induction on Seq generated by eseq, cons *)
by (res_inst_tac [("x","p")] (SEQ.Seq_Gen RS spec) 1);
by (rtac adm_flat 1);
by (rtac flat_lemma 1);
   (* case UU *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case eseq *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case cons *)
br impI 1;
by (forward_tac [c1] 1);
bd c2 1;
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
bd if_eq_TT 1;
      (* by cases x=y21 *);
by (res_inst_tac [("x1","x"),("y1","y21")] (eq_disj RS disjE) 1);
        (* case x=y21 *);
by (fast_tac HOL_cs 1);
       (* case x~=y21*)
br mp 1;
ba 1;
br conjI 1;
by (dres_inst_tac [("x2","x"),("y2","y21")] (weq RS c2 RS mp) 1);
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
by (res_inst_tac [("t","isin[x#y22]")] (hd(tl(ifte_thms)) RS subst) 1);
by (eres_inst_tac [("t","FF"),("s","x===y21")] (subst) 1);
ba 1;
by (fast_tac HOL_cs 1);
val SEQ_lem4=result();

val prems = goal SEQ.thy
"[|p~=UU;q~=UU;r~=UU|] ==> subset[p#q]=TT & subset[q#r]=TT --> subset[p#r]=TT";
by (cut_facts_tac prems 1);
(* by induction on Seq generated by eseq, cons *)
by (res_inst_tac [("x","p")] (SEQ.Seq_Gen RS spec) 1);
by (rtac adm_flat 1);
by (rtac flat_lemma 1);
   (* case UU *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case eseq *)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
   (* case cons *)
br impI 1;
by (forward_tac [c1] 1);
```

```
bd c2 1;
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
bd if_eq_TT 1;
      (* by cases x=y21 *);
by (res_inst_tac [("x1","x"),("y1","y21")] (eq_disj RS disjE) 1);
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
by (res_inst_tac [("s","isin[y21#r]")] (if_exp RS sym RS subst) 1);
by (res_inst_tac [("p1","q")] (SEQ_lem4 RS mp) 1);
by (REPEAT (fast_tac HOL_cs 1));
        (* case x~=y21*)
by (asm_full_simp_tac (S_ss addsimps (SEQ_Prg@SEQ_Data)) 1);
by (res_inst_tac [("s","isin[y21#r]")] (if_exp RS sym RS subst) 1);
by (res_inst_tac [("p1","q")] (SEQ_lem4 RS mp) 1);
by (REPEAT (fast_tac HOL_cs 1));
val SEQ_lem5 = result();
```

4.2.3 Lemmas for SEQ_ext

```
val has_x_total=conjI RS (SEQ_ext.SEQ_ext_Total RS c2 RS spec RS spec RS mp);
val add_x_total=conjI RS (SEQ_ext.SEQ_ext_Total RS c1 RS spec RS spec RS mp);
val add_x_strict1=disjI1 RS (SEQ_ext.SEQ_ext_Strict RS spec RS spec RS mp);
val add_x_strict2=disjI2 RS (SEQ_ext.SEQ_ext_Strict RS spec RS spec RS mp);

val SEQ_ext_Prg = [ SEQ_ext.construct_1, SEQ_ext.construct_2,
SEQ_ext.function_1 ];
val SEQ_ext_Data = [SEQ_ext.SEQ_ext_Def,SEQ_ext.SEQ_ext_Strict,
SEQ_ext.SEQ_ext_Total];
```

4.2.4 Lemmas for SEQ_REP

```
val is_Set_strict = SEQ_REP.SEQ_REP_Strict RS spec RS mp;
val is_Set_total = SEQ_REP.SEQ_REP_Total RS spec RS mp;
val SEQ_REP_Prg = [ SEQ_REP.rep1, SEQ_REP.rep2 ];
val SEQ_REP_Data = [SEQ_REP.SEQ_REP_Def,SEQ_REP.SEQ_REP_Strict,
SEQ_REP.SEQ_REP_Total];
val prems = goal SEQ_REP.thy
"[|x ~= UU; q ~= UU; is_Set[cons[x#q]]=TT|] ==> isin[x#q]=FF";
by (cut_facts_tac prems 1);
by (asm_full_simp_tac (SPECHOLCF_ss addsimps [SEQ_REP.rep2]) 1);
val rep_lemma1 = result();

val prems = goal SEQ_REP.thy
"[|x ~= UU; q ~= UU; is_Set[cons[x#q]]=TT|] ==> is_Set[q]=TT";
by (cut_facts_tac prems 1);
by (asm_full_simp_tac (SPECHOLCF_ss addsimps [SEQ_REP.rep2]) 1);
val rep_lemma2 = result();
```

```
val prems = goal SEQ_REP.thy
"[|x~=UU;q~=UU;is_Set[q]=TT|] ==> is_Set[add_x[x#q]]=TT";
by (cut_facts_tac prems 1);
by (asm_simp_tac (SPECHOLCF_ss addsimps [SEQ_ext.construct_2]) 1);
by (res_inst_tac [("p","isin[x#q]")] case_split 1);
bd isin_total 1;
ba 1;
by (fast_tac HOL_cs 1);
by (asm_simp_tac SPECHOLCF_ss 1);
by (asm_simp_tac SPECHOLCF_ss 1);
by (asm_simp_tac (SPECHOLCF_ss addsimps [SEQ_REP.rep2]) 1);
val is_set_add_x = result();
```

4.2.5 Lemmas for SEQ_EQ

```
val prems = goal SEQ_EQ.thy "[|q~=UU;is_Set[q]=TT|] ==>abs[q] ~= UU";
by (cut_facts_tac prems 1);
by (forward_tac [SEQ_EQ.partial] 1);
by (rewrite_tac [biimpl_def]);
bd c2 1;
br (UU_or_not_UU RS disjE) 1;
ba 2;
by (fast_tac HOL_cs 1);
val represent1 = result();

val prems = goal SEQ_EQ.thy "[|q~=UU;is_Set[q]=FF|] ==>abs[q] = UU";
by (cut_facts_tac prems 1);
bd FF_imp_not_TT 1;
by (forward_tac [SEQ_EQ.partial] 1);
by (rewrite_tac [biimpl_def]);
by (res_inst_tac [("x1","abs[q]")] (UU_or_not_UU RS disjE) 1);
by (fast_tac HOL_cs 1);
bd c1 1;
by (fast_tac HOL_cs 1);
val represent2 = result();

val prems = goal SEQ_EQ.thy "[|q~=UU;abs[q] ~= UU|] ==>is_Set[q]=TT";
by (cut_facts_tac prems 1);
by (forward_tac [SEQ_EQ.partial] 1);
by (rewrite_tac [biimpl_def]);
bd c1 1;
by (fast_tac HOL_cs 1);
val represent3 = result();
```

```
val abs_strict = SEQ_EQ.SEQ_EQ_Strict RS c1 RS spec RS mp;

val prems = goal SEQ_EQ.thy "abs[q]~=UU==>q~=UU";
by (cut_facts_tac prems 1);
br notnotD 1;
br contrapos 1;
br abs_strict 2;
ba 1;
by (fast_tac HOL_cs 1);
val abs_strict2 = result();

val eq_Set_strict = SEQ_EQ.SEQ_EQ_Strict RS c2 RS spec RS spec RS mp;

val SEQ_EQ_Prg = [SEQ_EQ.hom1,SEQ_EQ.hom2,SEQ_EQ.hom3,SEQ_EQ.eq];
val SEQ_EQ_Data=[SEQ_EQ.SEQ_EQ_Def,SEQ_EQ.SEQ_EQ_Strict,represent1,represent2];
```

4.2.6 Lemmas for SET_by_SEQ

```
goal SET_by_SEQ.thy
"(! (s::'a::eq Set_). (? (l::'a::eq Seq). (s = abs[l])))";
br SET_by_SEQ.Set_Gen 1;
br adm_flat 1;
br flat_lemma 1;
br exI 1;
br (abs_strict RS sym) 1;
br refl 1;
br exI 1;
br refl 1;
val Set_surjectiv = result();
val SET_by_SEQ_Prg = SEQ_ext_Prg @ SEQ_REP_Prg @ SEQ_EQ_Prg @ SEQ_Prg;
val SET_by_SEQ_Data = [SET_by_SEQ.instant] @ SEQ_ext_Data @ SEQ_REP_Data @
SEQ_EQ_Data @ SEQ_Data;
```

4.3 Proof of Set Axioms

This section contains the proofs for proof obligation 1.

4.3.1 Proof of Set_Gen

```
(* val prems = goal SET_by_SEQ.thy
"[|adm(P1);P1(UU);P1(empty);!!y21 y22.[|add[y21#y22] ~= UU;P1(y22)|] \
\                                   ==> P1(add[y21#y22])|] ==> \
\                    (! x1.P1(x1))"; *)
by (cut_facts_tac prems 1);
```

```
(* Induction Proof on Set generated by abs *)
br SET_by_SEQ.Set_Gen 1;
   (* Proofing preconditions *)
by (asm_simp_tac HOLCF_ss 1);
   (* case UU *)
ba 1;
   (* case abs(Seq): by induction on Seq generated by eseq, cons *)
br (SEQ.Seq_Gen RS spec) 1;
      (* Proofing preconditions *)
br adm_flat 1;
br flat_lemma 1;
      (* case UU *)
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
      (* case eseq *)
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
      (* case cons : by case destinction on the definedness of abs *)
by (res_inst_tac [("x1","abs[cons[y21#y22]]")] (UU_or_not_UU RS disjE) 1);
 (* UU *)
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
 (* ~UU : do definedness calculations *)
by (forward_tac [represent3] 1);
ba 1;
by (forward_tac [cons_strict] 1);
by (forward_tac [c1] 1);
bd c2 1;
by (forw_inst_tac [("x","y21"),("q","y22")] rep_lemma1 1);
ba 1;ba 1;
by (forw_inst_tac [("x","y21"),("q","y22")] rep_lemma2 1);
ba 1;ba 1;
    (* do necessary conversions *)
by (subgoal_tac "cons[y21#y22]=add_x[y21#y22]" 1);
by (asm_full_simp_tac (SPECHOLCF_ss addsimps(SET_by_SEQ_Prg@SET_by_SEQ_Data))2);
by (subgoal_tac "abs[cons[y21#y22]]=add[y21#abs[y22]]" 1);
br sym 2;
by (asm_simp_tac HOL_ss 2);
be (SEQ_EQ.hom2 RS mp) 2;
ba 2;ba 2;
by (dres_inst_tac [("s","cons[y21#y22]"),("t","add_x[y21#y22]")] sym 1);
by (asm_simp_tac HOL_ss 1);
    (* apply induction hyposis of Set generated by abs *)
br (hd(tl(tl(tl prems)))) 1;
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
val SET_by_SET_by_SEQ_Set_Gen = result();
```

4.3.2 Proof of Set_Exh

```
(* val prems = goal SET_by_SEQ.thy "x = UU | x = empty | \
\                    (? y1 y2. y1 ~= UU & y2 ~= UU & x = add[y1#y2])"; *)
(* by induction on Set generated by empty, add *)
br (SET_by_SET_by_SEQ_Set_Gen RS spec) 1;
   (* proving preconditions *)
br adm_flat 1;
br flat_lemma 1;
   (* case UU *)
by (asm_simp_tac HOLCF_ss 1);
   (* case empty *)
by (asm_simp_tac HOLCF_ss 1);
   (* case add *)
br disjI2 1;
br disjI2 1;
by (res_inst_tac [("x","y21")] exI 1);
by (res_inst_tac [("x","y22")] exI 1);
bd add_strict2 1;
by (fast_tac HOL_cs 1);
val SET_by_SET_by_SEQ_Set_Exh = result();
```

4.3.3 Proof of the Axiom set1

```
(* val prems = goal SET_by_SEQ.thy "x ~= UU ==> \
\                    not(has[x#empty] = TT)"; *)
by (cut_facts_tac prems 1);
(* simple rewrite proof *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
val SET_by_SET_by_SEQ_set1 = result();
```

4.3.4 Proof of the Axiom set2

```
(* val prems = goal SET_by_SEQ.thy "[|x ~= UU;s ~= UU|] ==> \
\                    has[x#add[x#s]] = TT"; *)
by (cut_facts_tac prems 1);
(* use the surjectivity of abs *)
by (res_inst_tac [("x1","s")] (Set_surjectiv RS spec RS exE) 1);
by (asm_simp_tac HOLCF_ss 1);
(* definedness *)
by (forw_inst_tac [("y","abs[xa]")] not_UU_trans 1);
ba 1;
by (forward_tac [abs_strict2] 1);
(* fulfil the restriction predicate is_Set to apply hom3 *)
by (subgoal_tac "is_Set[xa]=TT" 1);
by (forward_tac [SEQ_EQ.partial] 2);
```

```
by (rewrite_tac [biimpl_def]);
bd c1 2;
bd mp 2;
ba 2;ba 2;
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
(* definedness *)
by (forw_inst_tac [("x","x"),("q","xa")] cons_Total1 1);
ba 1;
by (forw_inst_tac [("x2","x"),("x1","xa")] isin_total 1);
ba 1;
(* case split due to the definition of add_x = if isin(x,q).. *)
by (res_inst_tac [("p","isin[x#xa]")] case_split 1);
   (* case UU *)
by (fast_tac HOL_cs 1);
   (* case TT *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
   (* case FF *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
val SET_by_SET_by_SEQ_set2 = result();
```

4.3.5 Proof of the Axiom set3

```
(* val prems = goal SET_by_SEQ.thy "[|x ~= UU;y ~= UU;s ~= UU|] ==> \
\                    not((x === y) = TT) --> has[x#add[y#s]] = has[x#s]"; *)
by (cut_facts_tac prems 1);
(* use the surjectivity of abs *)
by (res_inst_tac [("x1","s")] (Set_surjectiv RS spec RS exE) 1);
by (asm_simp_tac HOLCF_ss 1);
(* definedness *)
by (forw_inst_tac [("y","abs[xa]")] not_UU_trans 1);
ba 1;
by (forward_tac [abs_strict2] 1);
(* fulfil the restriction predicate is_Set to apply hom3 *)
by (subgoal_tac "is_Set[xa]=TT" 1);
by (forward_tac [SEQ_EQ.partial] 2);
by (rewrite_tac [biimpl_def]);
bd c1 2;
bd mp 2;
ba 2;ba 2;
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
by (forw_inst_tac [("x","y"),("q","xa")] cons_Total1 1);
ba 1;
by (forw_inst_tac [("x2","y"),("x1","xa")] isin_total 1);
ba 1;
(* case split due to the definition of add_x = if isin(x,q).. *)
```

```
by (res_inst_tac [("p","isin[y#xa]")] case_split 1);
   (* case UU *)
by (fast_tac HOL_cs 1);
   (* case TT *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
   (* case FF *)
br impI 1;
by (asm_simp_tac(S_ss addsimps([not_UU_weq]@SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
val SET_by_SET_by_SEQ_set3 = result();
```

4.3.6 Proof of the Axiom set4

```
(* val prems = goal SET_by_SEQ.thy "[|x ~= UU;s ~= UU|] ==> \
\                    add[x#add[x#s]] = add[x#s]"; *)
by (cut_facts_tac prems 1);
(* use the surjectivity of abs *)
by (res_inst_tac [("x1","s")] (Set_surjectiv RS spec RS exE) 1);
by (asm_simp_tac HOLCF_ss 1);
(* definedness *)
by (forw_inst_tac [("y","abs[xa]")] not_UU_trans 1);
ba 1;
by (forward_tac [abs_strict2] 1);
(* fulfil the restriction predicate is_Set to apply hom3 *)
by (subgoal_tac "is_Set[xa]=TT" 1);
by (forward_tac [SEQ_EQ.partial] 2);
by (rewrite_tac [biimpl_def]);
bd c1 2;
bd mp 2;
ba 2;ba 2;
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
(* definedness *)
by (forw_inst_tac [("x","x"),("q","xa")] cons_Total1 1);
ba 1;
by (forw_inst_tac [("x2","x"),("x1","xa")] isin_total 1);
ba 1;
(* case split due to the definition of add_x = if isin(x,q).. *)
by (res_inst_tac [("p","isin[x#xa]")] case_split 1);
   (* case UU *)
by (fast_tac HOL_cs 1);
   (* case TT : give here only necessary rules *)
by (asm_simp_tac (S_ss addsimps [SEQ_EQ.hom2,SEQ_ext.construct_2]) 1);
   (* case FF : here the knowledge of sequences is needed *)
by (asm_simp_tac (S_ss addsimps
   ([SEQ_lem1,SEQ_lem2]@SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
val SET_by_SET_by_SEQ_set4 = result();
```

4.3.7 Proof of the Axiom set5

```
(* val prems = goal SET_by_SEQ.thy "[|x ~= UU;y ~= UU;s ~= UU|] ==> \
\                    add[x#add[y#s]] = add[y#add[x#s]]"; *)
by (cut_facts_tac prems 1);
by (res_inst_tac [("x1","s")] (Set_surjectiv RS spec RS exE) 1);
by (asm_simp_tac HOLCF_ss 1);
(* definedness *)
by (forw_inst_tac [("x","x"),("y","y")] total_weq 1);
ba 1;
by (forw_inst_tac [("y","abs[xa]")] not_UU_trans 1);
ba 1;
by (forward_tac [abs_strict2] 1);
(* fulfil the restriction predicate is_Set to apply hom2 *)
by (subgoal_tac "is_Set[xa]=TT" 1);
by (forward_tac [SEQ_EQ.partial] 2);
by (rewrite_tac [biimpl_def]);
bd c1 2;
bd mp 2;
ba 2;ba 2;
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
(* definedness *)
by (forw_inst_tac [("x","x"),("q","xa")] cons_Total1 1);
ba 1;
by (forw_inst_tac [("x2","x"),("x1","xa")] isin_total 1);
ba 1;
by (forw_inst_tac [("x","y"),("q","xa")] cons_Total1 1);
ba 1;
by (forw_inst_tac [("x2","y"),("x1","xa")] isin_total 1);
ba 1;
(* fulfil the restriction predicate is_Set to apply hom2 for add(y,s)*)
by (subgoal_tac "is_Set[If isin[y#xa] then xa else cons[y#xa] fi]=TT" 1);
   (* by cases isin(y,xa) *)
by (res_inst_tac [("p","isin[y#xa]")] case_split 2);
      (* case UU *)
by (fast_tac HOL_cs 2);
      (* case TT *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
      (* case FF *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
(* fulfil the restriction predicate is_Set to apply hom2 for add(x,s)*)
by (subgoal_tac "is_Set[If isin[x#xa] then xa else cons[x#xa] fi]=TT" 1);
   (* by cases isin(x,xa) *)
by (res_inst_tac [("p","isin[x#xa]")] case_split 2);
      (* case UU *)
by (fast_tac HOL_cs 2);
```

```
      (* case TT *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
      (* case FF *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
by (asm_simp_tac (S_ss addsimps ([if_total]@SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
   (* by cases isin(x,xa) *)
by (res_inst_tac [("p","isin[x#xa]")] case_split 1);
   (* case UU *)
by (fast_tac HOL_cs 1);
   (* case TT: by cases isin(y,xa) *)
by (res_inst_tac [("p","isin[y#xa]")] case_split 1);
      (* case UU *)
by (fast_tac HOL_cs 1);
      (* case TT *)
by (asm_simp_tac (S_ss addsimps ([if_total,SEQ_lem1,SEQ_lem2]@
SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
      (* case FF *)
by (asm_simp_tac (S_ss addsimps ([if_total,if_eq,SEQ_lem1,SEQ_lem2]@
SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
   (* case FF: by cases isin(y,xa) *)
by (res_inst_tac [("p","isin[y#xa]")] case_split 1);
      (* case UU *)
by (fast_tac HOL_cs 1);
      (* case TT *)
by (asm_simp_tac (S_ss addsimps ([if_total,if_eq,SEQ_lem1,SEQ_lem2]@
SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
      (* case FF *)
by (asm_simp_tac (S_ss addsimps ([if_total,if_eq,SEQ_lem1,SEQ_lem2]@
SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
         (* by cases x===y *)
by (res_inst_tac [("p","x===y")] case_split 1);
 (* case UU *)
by (fast_tac HOL_cs 1);
 (* case TT *)
by (asm_simp_tac (S_ss addsimps ([if_total,if_eq,weq_sym,SEQ_lem1,SEQ_lem2]@
SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
 (* case FF *)
by (asm_simp_tac (S_ss addsimps ([if_total,if_eq,weq_sym])) 1);
    (* definedness reasoning *)
by (forw_inst_tac [("x","y"),("q","cons[y#xa]")] cons_Total1 1);
ba 1;
by (forw_inst_tac [("x","y"),("q","cons[x#xa]")] cons_Total1 1);
ba 1;
    (* calculating the premises for {instant} *)
by (subgoal_tac "is_Set[cons[x#xa]]=TT" 1);
```

```
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
by (subgoal_tac "is_Set[cons[y#xa]]=TT" 1);
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
by (subgoal_tac "is_Set[cons[x#cons[y#xa]]]=TT" 1);
by (asm_full_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
by (subgoal_tac "is_Set[cons[y#cons[x#xa]]]=TT" 1);
by (asm_simp_tac (S_ss addsimps ([weq_sym]@SET_by_SEQ_Prg@SET_by_SEQ_Data)) 2);
    (* apply instant *)
by (asm_simp_tac (S_ss addsimps ([if_total,if_eq,SEQ_lem1,SEQ_lem2]@
SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
val SET_by_SET_by_SEQ_set5 = result();
```

4.3.8 Proof of the Axiom SET_Def

```
(* val prems = goal SET_by_SEQ.thy "empty ~= UU & add ~= UU & has ~= UU"; *)
br SIG_SET.SIG_SET_Def 1;
val SET_by_SET_by_SEQ_SET_Def = result();
```

4.3.9 Proof of the Axiom SET_Strict

```
(* val prems = goal SET_by_SEQ.thy
"(! x1 x2. (x1 = UU) | (x2 = UU) --> (add[x1#x2] = UU))"; *)
br add_strict1 1;
val SET_by_SET_by_SEQ_SET_Strict = result();
```

4.3.10 Proof of the Axiom SET_Total

```
(* goal comes from the frame
val prems = goal SET_by_SEQ.thy
"(! x1 x2. (x1 ~= UU) & (x2 ~= UU) --> (has[x1#x2] ~= UU)) &\
\ (! x1 x2. (x1 ~= UU) & (x2 ~= UU) --> (add[x1#x2] ~= UU))"; *)
br conjI 1;
(* first total axiom: has *)
br allI 1;
br allI 1;
   (* use exhaustiveness off abs *)
by (res_inst_tac [("x1","x2")] (Set_surjectiv RS spec RS exE) 1);
by (asm_simp_tac HOLCF_ss 1);
br impI 1;
   (* definedness reasoning *)
by (forward_tac [conjunct2] 1);
by (forward_tac [conjunct1] 1);
bd abs_strict2 1;
   (* fulfil the restriction predicate is_Set to apply hom3 *)
by (subgoal_tac "is_Set[x]=TT" 1);
by (forward_tac [SEQ_EQ.partial] 2);
```

```
by (rewrite_tac [biimpl_def]);
by (forward_tac [conjunct2] 2);
by (forw_inst_tac [("P","(abs[x] ~= UU --> is_Set[x] = TT)")] conjunct1 2);
by (asm_full_simp_tac HOLCF_ss 2);
   (* apply hom3 *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
(* next total axiom add *)
br allI 1;
br allI 1;
   (* use exhaustiveness off abs *)
by (res_inst_tac [("x1","x2")] (Set_surjectiv RS spec RS exE) 1);
by (asm_simp_tac HOLCF_ss 1);
br impI 1;
   (* definedness reasoning *)
by (forward_tac [conjunct2] 1);
bd c1 1;
by (forward_tac [abs_strict2] 1);
   (* fulfil the restriction predicate is_Set to apply hom2 *)
by (subgoal_tac "is_Set[x]=TT" 1);
by (forward_tac [SEQ_EQ.partial] 2);
by (rewrite_tac [biimpl_def]);
by (forward_tac [conjunct2] 2);
by (forw_inst_tac [("P","(abs[x] ~= UU --> is_Set[x] = TT)")] conjunct1 2);
by (asm_full_simp_tac HOLCF_ss 2);
   (* apply hom2 *)
by (asm_simp_tac (S_ss addsimps (SET_by_SEQ_Prg@SET_by_SEQ_Data)) 1);
   (* show that the result is defined i.e. is a Set *)
by (forw_inst_tac [("x","x1"),("q","x")] is_set_add_x 1);
ba 1; ba 1;
   (* definedness reasoning *)
by (forw_inst_tac [("x2","x1"),("x1","x")] isin_total 1);
ba 1;
by (forw_inst_tac [("x","x1"),("q","x")] cons_Total1 1);
ba 1;
by (forw_inst_tac [("b","isin[x1#x]"),("x","x"),("y","cons[x1#x]")] if_total 1);
ba 1;ba 1;
by (asm_full_simp_tac (S_ss addsimps [SEQ_EQ.partial]@SET_by_SEQ_Prg@
   SET_by_SEQ_Data) 1);
(* val SET_by_SET_by_SEQ_SET_Total = result();
result is in the frame *)
```

4.4 Proof of Congruence Relation

This section contains the proofs for proof obligation 3.

4.4.1 Proof of the Axiom sym

```
(* goal comes from the frame
val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU|] ==> \
\                    is_Set[p] = TT & is_Set[q] = TT --> \
\                    eq_Set[p#q] = eq_Set[q#p]"; *)
by (cut_facts_tac prems 1);
br impI 1;
by (asm_simp_tac (SPECHOLCF_ss addsimps [SEQ_EQ.eq,parand_sym]) 1);
val SET_cong_by_SEQ_EQ_sym = result();
```

4.4.2 Proof of the Axiom refl

```
(* goal comes from the frame
val prems = goal SEQ_EQ.thy "p ~= UU ==> \
\                    is_Set[p] = TT --> eq_Set[p#p] = TT"; *)
by (cut_facts_tac prems 1);
br impI 1;
by (asm_simp_tac (SPECHOLCF_ss addsimps [SEQ_EQ.eq,SEQ_lem2]) 1);
val SET_cong_by_SEQ_EQ_refl = result();
```

4.4.3 Proof of the Axiom trans

```
(* goal comes from the frame
val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU;r ~= UU|] ==> \
\                    is_Set[p] = TT & \
\                    is_Set[q] = TT & \
\                    is_Set[r] = TT & eq_Set[p#q] = TT & eq_Set[q#r] = TT\
\                     --> eq_Set[p#r] = TT"; *)
by (cut_facts_tac prems 1);
br impI 1;
(* apply definition of eq *)
by (asm_full_simp_tac (SPECHOLCF_ss addsimps [SEQ_EQ.eq]) 1);
(* skip & split preconditions *)
bd c2 1;
bd c2 1;
bd c2 1;
by (forward_tac [c1] 1);
bd c2 1;
bd (parand_TT RS mp) 1;
bd (parand_TT RS mp) 1;
by (forward_tac [c1] 1);
bd c2 1;
by (forward_tac [c1] 1);
bd c2 1;
(* split into two subgoals *)
```

```
br (parand_TT2 RS mp) 1;
br conjI 1;
   (* solve first by trans-lemma *)
by (res_inst_tac [("q1","q")] (SEQ_lem5 RS mp) 1);
by (REPEAT (atac 1));
by (fast_tac HOL_cs 1);
   (* solve second by trans-lemma *)
by (res_inst_tac [("q1","q")] (SEQ_lem5 RS mp) 1);
by (REPEAT (atac 1));
by (fast_tac HOL_cs 1);
val SET_cong_by_SEQ_EQ_trans = result();
```

4.4.4 Proof of the Axiom cong_add_x

```
(* goal comes from the frame
val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU;x ~= UU|] ==> \
\                    is_Set[p] = TT & is_Set[q] = TT & eq_Set[p#q] = TT --> \
\                    eq_Set[add_x[x#p]#add_x[x#q]] = TT"; *)
by (cut_facts_tac prems 1);
br impI 1;
(* splitting preconditions *)
by (forward_tac [c1] 1);
bd c2 1;
by (forward_tac [c1] 1);
bd c2 1;
(* definedness reasoning *)
by (forw_inst_tac [("x2","x"),("x1","p")] add_x_total 1);
ba 1;
by (forw_inst_tac [("x2","x"),("x1","q")] add_x_total 1);
ba 1;
(* unfold definition of eq_Set *)
by (asm_full_simp_tac (SPECHOLCF_ss addsimps [SEQ_EQ.eq]) 1);
by (forw_inst_tac [("x","x"),("q","p")] is_set_add_x 1);
by (REPEAT (atac 1));
by (forw_inst_tac [("x","x"),("q","q")] is_set_add_x 1);
by (REPEAT (atac 1));
(* prepare definedness for cases *)
by (forw_inst_tac [("x2","x"),("x1","p")] isin_total 1);
ba 1;
by (forw_inst_tac [("x2","x"),("x1","q")] isin_total 1);
ba 1;
bd (parand_TT RS mp) 1;
by (forward_tac [c1] 1);
bd c2 1;
(* apply rules of add_x *)
```

```
by (asm_simp_tac (SPECHOLCF_ss addsimps SEQ_ext_Prg) 1);
(* by cases isin(x,p) *)
by (res_inst_tac [("p","isin[x#p]")] case_split 1);
   (* case UU *)
by (fast_tac HOL_cs 1);
   (* case isin(x,p)=TT : by cases isin(x,q) *)
by (res_inst_tac [("p","isin[x#q]")] case_split 1);
      (* case UU *)
by (fast_tac HOL_cs 1);
      (* case isin(x,p)=TT *)
by (asm_simp_tac (SPECHOLCF_ss addsimps ([cons_Total1,SEQ_lem1]@SEQ_Prg)) 1);
      (* case isin(x,p)=FF *)
by (asm_simp_tac (SPECHOLCF_ss addsimps ([cons_Total1,SEQ_lem1]@SEQ_Prg)) 1);
   (* case isin(x,p)=FF : by cases isin(x,q) *)
by (res_inst_tac [("p","isin[x#q]")] case_split 1);
      (* case UU *)
by (fast_tac HOL_cs 1);
      (* case isin(x,p)=TT *)
by (asm_simp_tac (SPECHOLCF_ss addsimps ([cons_Total1,SEQ_lem1]@SEQ_Prg)) 1);
      (* case isin(x,p)=FF *)
by (asm_simp_tac (SPECHOLCF_ss addsimps ([cons_Total1,SEQ_lem1]@SEQ_Prg)) 1);
val SET_cong_by_SEQ_EQ_cong_add_x = result();
```

4.4.5 Proof of the Axiom cong_has_x

```
(* goal comes from the frame
val prems = goal SEQ_EQ.thy "[|p ~= UU;q ~= UU;x ~= UU|] ==> \
\                    is_Set[p] = TT & is_Set[q] = TT & eq_Set[p#q] = TT --> \
\                    has_x[x#p] = has_x[x#q]"; *)
by (cut_facts_tac prems 1);
br impI 1;
(* splitting preconditions *)
by (forward_tac [c1] 1);
bd c2 1;
by (forward_tac [c1] 1);
bd c2 1;
(* definedness reasoning *)
by (forw_inst_tac [("x2","x"),("x1","p")] add_x_total 1);
ba 1;
by (forw_inst_tac [("x2","x"),("x1","q")] add_x_total 1);
ba 1;
(* unfold definition of eq_Set *)
by (asm_full_simp_tac (SPECHOLCF_ss addsimps [SEQ_EQ.eq]) 1);
by (forw_inst_tac [("x","x"),("q","p")] is_set_add_x 1);
by (REPEAT (atac 1));
```

```
by (forw_inst_tac [("x","x"),("q","q")] is_set_add_x 1);
by (REPEAT (atac 1));
(* prepare definedness for cases *)
by (forw_inst_tac [("x2","x"),("x1","p")] isin_total 1);
ba 1;
by (forw_inst_tac [("x2","x"),("x1","q")] isin_total 1);
ba 1;
bd (parand_TT RS mp) 1;
by (forward_tac [c1] 1);
bd c2 1;
(* apply rules of add_x *)
by (asm_simp_tac (SPECHOLCF_ss addsimps SEQ_ext_Prg) 1);
(* by cases isin(x,q) *)
by (res_inst_tac [("p","isin[x#q]")] case_split 1);
   (* case UU *)
by (fast_tac HOL_cs 1);
   (* case TT *)
by (subgoal_tac "isin[x#p]=TT" 1);
by (res_inst_tac [("p1","q"),("x1","x")] (SEQ_lem4 RS mp) 2);
by (REPEAT (atac 2));
by (asm_simp_tac SPECHOLCF_ss 1);
by (asm_simp_tac SPECHOLCF_ss 1);
   (* case FF *)
by (subgoal_tac "isin[x#p]=FF" 1);
by (asm_simp_tac SPECHOLCF_ss 1);
by (res_inst_tac [("p1","q")] (SEQ_lem3 RS mp) 1);
by (REPEAT (atac 1));
by (fast_tac HOL_cs 1);
val SET_cong_by_SEQ_EQ_cong_has_x = result();
```

4.5 Proof of Invariance

This section contains the proofs for proof obligation 2.

4.5.1 Proof of the Axiom invar1

```
(* goal comes from the frame
 val prems = goal SEQ_REP.thy "is_Set[empty_x] = TT"; *)
by (cut_facts_tac [def_eseq] 1);
by (asm_simp_tac (SPECHOLCF_ss addsimps ([cons_Total1,isin_total,def_eseq,
 weq_refl,if_total]@SEQ_Prg@SEQ_ext_Prg @ SEQ_REP_Prg)) 1);
val SET_inv_by_SEQ_REP_invar1 = result();
```

4.5.2 Proof of the Axiom invar2

```
(* goal comes from the frame
 val prems = goal SEQ_REP.thy "[|q ~= UU;x ~= UU|] ==> \
\                    is_Set[q] = TT --> is_Set[add_x[x#q]] = TT"; *)
by (cut_facts_tac prems 1);
br impI 1;
(* important lemma is proved in SEQ_REP.ML *)
be is_set_add_x 1;
ba 1;ba 1;
val SET_inv_by_SEQ_REP_invar2 = result();
```

I would like to thank Prof. Manfred Broy for the interesting topic he gave me and Christian Prehofer and Franz Regensburger for comments on a draft version of this report.

References

[BFG+93a] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Secification Language SPECTRUM. An Informal Introduction. Version 1.0. Part I. Technical Report TUM-I9311, Technische Universität München. Institut für Informatik, May 1993.

[BFG+93b] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Secification Language SPECTRUM. An Informal Introduction. Version 1.0. Part II. Technical Report TUM-I9312, Technische Universität München. Institut für Informatik, May 1993.

[BW93] M. Broy and M. Wirsing. Korrekte Software: Vom Experiment zur Anwendung. In Horst Reichel, editor, *GI Informatik aktuell. Informatik, Wirtschaft, Gesellschaft*, pages 29–43. Springer-Verlag, 1993.

[EKMP82] H. Ehrig, H.-J. Kreowski, B. Mahr, and P. Padawitz. Algebraic implementation of abstract data types. *Theoretical Computer Science*, 20:209–263, 1982.

[HJW92] P. Hudak, S. Peyton Jones, and P. Wadler, editors. *Report on the Programming Language Haskell, A Non-strict Purely Functional Language (Version 1.2)*. ACM SIGPLAN Notices, May 1992.

[Jon93] M. P. Jones. *An Introduction to Gofer*, August 1993.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer, 1994.

[PBDD94] P. Pepper, R. Betschko, S. Dick, and K. Didrich. Realizing sets by hash tables: How to do it in korso. Technical Report 94-30, TU Berlin, August 1994.

[Reg94] Franz Regensburger. *HOLCF: Eine konservative Erweiterung von HOL um LCF.* PhD thesis, Technische Universität München, 1994.

[Slo95] O. Slotosch. Verification Conditions for the Change of Data Structures. paper submitted, 1995.