Spivey's BirthdayBook Example in HOL-Z

May 28, 2008

Contents

1 The Specification

This document cantains specification and analysis of Spivey's classical BirthdayBook example. It is intended to demonstrate the use of the ZETA-frontend to write and typecheck specifications in Z, and the Isabelle backend that allows for stating and proving proof obligations over this specification.

1.1 Basic Datatypes and State Schemas

section *BBSpec*

$[NAME, DATE]$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{BirthdayBook}} \\ \hline
known : \mathbb{P}\, NAME \\
birthday : NAME \pfun DATE \\ \hline
known = \operatorname{dom} birthday \\ \hline
\end{array}$$

1.2 Operations

$InitBirthdayBook == [BirthdayBook \mid known = \varnothing]$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AddBirthday}} \\ \hline
\Delta BirthdayBook \\
name? : NAME;\ date? : DATE \\ \hline
name? \notin known \\
birthday' = birthday \cup \{name? \mapsto date?\} \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{FindBirthday}} \\ \hline
\Xi BirthdayBook \\
name? : NAME;\ date! : DATE \\ \hline
name? \in known \\
date! = birthday(name?) \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Remind}} \\ \hline
\Xi BirthdayBook \\
today? : DATE;\ cards! : \mathbb{P}\, NAME \\ \hline
cards! = \{n : NAME \mid n \in known;\ birthday(n) = today?\} \\ \hline
\end{array}$$

1.3 Strengthening the Specification

$REPORT ::= ok \mid already\_known \mid not\_known$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Success}} \\ \hline
result! : REPORT \\ \hline
result! = ok \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AlreadyKnown}} \\ \hline
\Xi BirthdayBook \\
name? : NAME \\
result! : REPORT \\ \hline
name? \in known \\
result! = already_known \\ \hline
\end{array}$$

$$RAddBirthday == (AddBirthday \land Success) \lor AlreadyKnown$$

1.4 Implementing the Birthday Book

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{BirthdayBook1} \\
names : \mathbb{N} \rightarrow NAME \\
dates : \mathbb{N} \rightarrow DATE \\
hwm : \mathbb{N} \\
\hline
\forall\, i, j : 1 \mathbin{..} hwm \bullet i \neq j \Rightarrow names(i) \neq names(j) \\
\hline
\end{array}
$$

$$InitBirthdayBook1 == [BirthdayBook1 \mid hwm = 0]$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{Abs} \\
BirthdayBook \\
BirthdayBook1 \\
\hline
known = \{ i : 1 \mathbin{..} hwm \bullet names(i) \} \\
\forall\, i : 1 \mathbin{..} hwm \bullet birthday(names(i)) = dates(i) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{AddBirthday1} \\
\Delta BirthdayBook1 \\
name? : NAME \\
date? : DATE \\
\hline
\forall\, i : 1 \mathbin{..} hwm \bullet name? \neq names(i) \\
hwm' = hwm + 1 \\
names' = names \oplus \{ hwm' \mapsto name? \} \\
dates' = dates \oplus \{ hwm' \mapsto date? \} \\
\hline
\end{array}
$$

1.5 Specification

1.6 Stating Conjectures

At times, the designer of a specification might want to state a certain property that he has in mind when writing the specification document. Such properties can be stated as *conjecture*. ZETA can type-check them and export them to HOL-Z; the latter will consider a conjecture as definition of an internal constant symbol. For proving the conjecture, one states a lemma in the analysis that this internal constant symbol is actually equivalent to *True*.

The statement of a conjecture is simply done by:

$$\forall\, BirthdayBook;\ BirthdayBook1;\ name? : NAME;\ date? : DATE \bullet$$
$$name? \notin known \land (known = \{i : 1 \mathinner{\ldotp\ldotp} hwm \bullet names(i)\})$$
$$\Rightarrow (\forall\, i : 1 \mathinner{\ldotp\ldotp} hwm \bullet name? \neq names(i))$$

1.7 Conclusion

In the following, we discuss two versions of analysis: one based on relational refinement, another on functional refinement, which leads to simpler proof. See corresponding `BBSpec.thy` file for the relational refinement, and `BBSpec_Functional.thy` for the functional refinement.

The following ISAR command starts a new Isabelle theory based on Z, including all libraries and setups.

2 Import, Inspection, and Basic Analysis of the Specification

theory *BBSpec*
imports *Z*

begin

2.1 Loading a ZETA-unit and the Consequences

The following HOL-Z toplevel commands allows for loadong directly the output of the ZETA format, in this case, we load the previous specification `BBSpec` presented in the previous section.

load_holz *"BBSpec"*

thm *AddBirthday_def*

This leads to a new state of the proof environment where all sorts of elements of the specification were bound to ISAR names and can therefore be referenced in future proofs.

A guided tour through the generated definitions looks as follows: For abstract types, constants destribing the type sets exist:

thm *NAME_def*

Data types result in a number of simplification rules

thm *BBSpec.REPORT.simps*

Schema (both states and operation schemas) were converted to constant definitions denoting the "set of records", i.e. their value

thm *BirthdayBook_def*

thm *Remind_def*
InitBirthdayBook_def
AddBirthday_def
RAddBirthday_def

Schemas, axiomatic definitions and conjectures were collected into the variables:

thm *SCHEMAS*
thm *AXDEFS*
thm *CONJECTURES*

2.2 Analysis I: Proving Conjectures

Example:

```
lemma conjecture_0_proof : "conjecture_0"
by(unfold conjecture_0_def,zstrip,
   zunfold BirthdayBook_def BirthdayBook1_def, auto simp: Z2HOL)
```

Of course, the conjecture can also be stated in the analysis document (the .thy-file) directly.

```
zlemma PO_refine_1_AddBirthday_simple :
"(%A BirthdayBook @ (%A BirthdayBook1 @
    (∀ name? ∈ NAME. ∀ date? ∈ DATE.
      ((name? ∉ known ∧ known = {n. ∃ i ∈ 1..hwm. n = (names%^i)})
      ⟶
      (∀ i ∈ 1..hwm. name? ≠ (names %^ i)))
 )))"
by(zstrip,auto)
```

Note that the input of the formula must be done with the zlemma command since this supports HOL-Z syntax (and not just HOL syntax).

2.3 Analysis II: Checking Consistency

Now we turn to the statement of analysis judgements and refinement judgements. Both lead to the generation of proof obligations.

The refinement package provides a number of generic toplevel commands for these analysis judgements and, moreover, methods to inspect and discharge proof obligations.
Such method-specific support helps a lot to improve critical review of specifications ("is this really what you want to specifiy?"), as well as a basis for specific tactics.

Generic inspection commands are:

list_po
check_po

. . . which provide functionality for listing pending proof obligations and also checks that they have been discharged. At the moment, both commands are nops since no proof obligation has been generated.

The first analytic judgement statement commands are:

gen_state_cc *BirthdayBook*
gen_state_cc *BirthdayBook1*

. . . which lead to the generation of the proof obligations `BBSpec.ccState_BirthdayBook_1` and `BBSpec.ccState_BirthdayBook1_1`. They state that the state schemas (representing invariants) should be satisfiable. With the generic command:

show_po *BBSpec.ccState_BirthdayBook1_1*

As an example, we will dicharge this proof oblifation by providing a proof for it.

po *"BBSpec.ccState_BirthdayBook1_1"*

This means that we have to show:

1. ∃ *BirthdayBook1* • *True*

Obvioulsly, we have to eliminate the schema quantifier by an respective introduction tactic for the schema calulus:

apply(*zintro_sch_ex,clarify,(rule refl)+*)
apply(*zunfold BirthdayBook1_def*)
apply(*auto simp: Z2HOL*)
apply(*rule_tac [3] ZInteg.zero_is_natural, simp_all*)
apply(*rule_tac f="λ x. arbitrary"* **in** *lambda_total1,simp*)+
discharged

. . . we can display the precise form of proof obligation.

Moreover, we generate proof obligations that assure that an operation schema is in fact implementable or "non-blocking", in the sense that there is a (not necessarily computable) function mapping pre-state and input to a post-state and output.

gen_op_cc *AddBirthday*
gen_op_cc *AddBirthday1*

show_po *BBSpec.ccOp_AddBirthday1_1*

Note that the listing of now active proof obligations can be parameterized by filters excluding certain proof-obligation classes (more filtering functions are desirable, but currently not implemented):

list_po except *ccOp*

end

3 Analysis by Functional Refinement

theory *Fun_Refinement*
imports *BBSpec*

begin

3.1 Setting up the functional refinement

Now we set the default abstraction relation in the refinement package; this setting is a pre-requisite to the future generation of refinement related proof obligations.

set_abs *"Abs"[***functional***]*

At this place, we add the directive "[functional]" in order to indicate that the abstraction relation is in fact a function.

This results in the additional proof obligation that the abstraction relation is in fact a function, but leads to simpler proof obligations over this abstraction relation later.

Now comes the core of proof obligation generation based on the Forward-Simulation Refinement Method for Z (see Spivey's Book or "Using Z")

```
refine_init InitBirthdayBook InitBirthdayBook1
refine_op   AddBirthday      AddBirthday1

show_po Fun_Refinement.fwRefinementInit_BirthdayBook_1
show_po Fun_Refinement.fwRefinementOp_AddBirthday_2
list_po

show_po BBSpec.ccOp_AddBirthday_1
        BBSpec.ccOp_AddBirthday1_1
        BBSpec.ccState_BirthdayBook_1
        BBSpec.ccState_BirthdayBook1_1
        Fun_Refinement.fwRefinementFunctional_Abs_1
        Fun_Refinement.fwRefinementOp_AddBirthday_1
        Fun_Refinement.fwRefinementOp_AddBirthday_2
        Fun_Refinement.fwRefinementInit_BirthdayBook_1
```

We perform a final check of the proof obligations; however, we filter out certain classes of proof-obligations.

```
check_po except ccOp ccState fwRefinementFunctional
                fwRefinementOp fwRefinementInit
```

In this example, we actually filter out **all** obligations.

In contrast, the toplevel command:

```
check_po
```

would result in a failure:

```
*** There are 7 unproven proof-obligations (can not ignore!).
*** Check failed.
*** At command "check_po".
```

3.2 Proof of Functionality of the Abstraction Relation

```
lemma lemma1:
 "⟦ BirthdayBook1 (dates, hwm, names); i : ( 1 .. hwm); ia : ( 1 .. hwm);
    names %^ i = names %^ ia ⟧
  ⟹ dates %^ i = dates %^ ia"
apply(zunfold BirthdayBook1_def, simp add: Z2HOL, clarify)
apply(case_tac "i=ia",auto)
done
```

lemma *lemma2:*
```
"(a : (rel_appl names) ‘ ( 1 .. hwm)) = (EX i: 1..hwm. names %^ i = a)"
```
apply*(auto simp:Z2HOL)*
done

po *"Fun_Refinement.fwRefinementFunctional_Abs_1"*
apply*(zstrip)*
apply*(simp add:Z2HOL Ex1_def)*
apply*(rule_tac x="{(x,y). EX i:(1 .. hwm). x = names %^ i ∧ y = dates %^ i}"* **in** *exI)*
apply*(rule_tac x="(rel_appl names) ‘ (asSet(%i. i : (1 .. hwm)))"* **in** *exI)*
apply*(zunfold Abs_def BirthdayBook_def)*
apply*(simp add: Z2HOL Ex1_def)*
apply*(safe, simp_all)*
apply*(simp only:pfun_def rel_def, auto intro!: lemma1)+*
apply*(subst ZFun.beta_apply_pfun[of _ NAME DATE])*
prefer *3*
apply*(rule refl)*
apply*(auto)*
apply*(rule pfunI)*
apply*(simp add:rel_def)*
apply *auto*
apply*(auto intro!: lemma1)*
prefer *2*
apply*(rule_tac t="dates %^ i"* **in** *subst)*
prefer *2*
apply*(erule ZFun.rel_apply_in_rel,auto)*
apply*(drule_tac x=aa* **in** *eqset_imp_iff,auto)*
apply*(rule_tac x=x* **in** *bexI, auto)*
discharged

3.3 Proofs of the Init-Condition of the Refinement

po *Fun_Refinement.fwRefinementInit_BirthdayBook_1*

To show:

1. ∀ *BirthdayBook* • (∀ *BirthdayBook1* • (*InitBirthdayBook1* ∧ *Abs* ⟶ *InitBirthdayBook*))

We perform structural simplification by eliminating Schema-Calculus-Constructs.

apply *zstrip*

Now we follow the brute force approach: unfolding all schema definitions ...

apply*(zunfold InitBirthdayBook1_def InitBirthdayBook_def Abs_def BirthdayBook_def)*

Conversion to plain HOL and using the simplifier just finds the witness $(\{\}, \{\})$ automatically.

apply*(simp add: Z2HOL)*
discharged

3.4 Proof of the First Operation-Refinement-Condition

In the following, we introduce three lemmas that allow the reduction of the first refinement condition to the simplified version above.

We provide three auxilliary lemmas.

First: the syntactic precondition over leagal states implies the semantic precondition for AddBirthday1:

zlemma *lemma3 :*
"BirthdayBook1 ∧ (∀ i∈1..hwm. name? ≠ (names %^ i)) ⟶ pre AddBirthday1"

We start with elementary Z-logical massage:

apply*(zstrip, zintro_pre AddBirthday1_def)*
apply*(simp add: DECL_def DELTA_def, rule conjI)*

. . . and split the declarations from body.

In particular, we take the prescribed successor state and propagate it in proof.

apply*(rule_tac [2] conjI | rule_tac [2] refl)+*

. . . proves that the successor state fulfills state invariant.

apply*(zunfold BirthdayBook1_def)*
apply*(simp add: Ball_def maplet_def zpred_def, auto)*

. . . does its best to make it simpler

auto reduces the proof to a "proof by contradiction" scheme ... Mostly, it has to do with

```
(names (+) {(hwm + 1, name?)}) %^ i =
(names (+) {(hwm + 1, name?)}) %^ j;
```

while we know that `names %^ i ~= names %^ j`. We bring this goal in the end of the assumption list:

apply*(rotate_tac 1)*

Now comes the proof idea: We split up a case-distinction tree for the cases that i and j refer to the new element . . . `rotate_tac` brings these clauses in front in the assumption list and makes them visible for the rewriter.

apply*(case_tac [1] "x=(hwm+1)")*
apply*(case_tac [1] "xa=(hwm+1)")*
apply*(case_tac [3] "xa=(hwm+1)")*

Since we know already that `i~=j` and `ALL i: #1 .. hwm. name? ~= names %^ i`, these four cases can be reduced ad absurdum.

apply*(auto simp: zpred_def)*
done

Second: The semantic precondition of the abstract operation implies syntactic precondition

ML*{*show_full_sem:=true*}*
zlemma *lemma4 : "pre AddBirthday ⟶ name? ∉ known"*
by*(zstrip,zelim_pre,zunfold AddBirthday_def,auto)*

here comes a structural proof for the first main goal: use the above three Z-lemmas in order to reduce the main goal to the simplified version of it above. The technique applies the stripS converter to bring Z-lemmas on-thy-fly into HOL form and introduce them into the proof by Isabelle standard tactics.

po *Fun_Refinement.fwRefinementOp_AddBirthday_1*

To show:

1. ∀ asSet BirthdayBook • (SB ''birthday'' ~~> birthday, ''known'' ~~> known. ∀ asSet BirthdayBook1 • (SB ''dates'' ~~> dates, ''hwm'' ~~> hwm, ''names'' ~~> names. ∀date? name?. pre (SB ''birthday‘'' ~~> birthday‘, ''known‘'' ~~> known‘. SNAME AddBirthday (birthday, birthday‘, date?, known, known‘, name?)) ∧ SNAME Abs (birthday, dates, hwm, known, names) ⟶ pre (SB ''dates‘'' ~~> dates‘, ''hwm‘'' ~~> hwm‘, ''names‘'' ~~> names‘. SNAME AddBirthday1 (date?, dates, dates‘, hwm, hwm‘, name?, names, names‘))))

After structural normalization:

apply*(zstrip, clarify)*

. . . we use the lemmas 1) to 3) by weakening assumptions and reducing conclusions.

apply*(zrule lemma3,zdrule lemma4,zdrule Abs_def[zpred [1]])*
apply*(auto simp: Z2HOL)*
discharged

3.5 Proof of the Second Operation-Refinement-Condition

To establish the second refinement condition, we need two auxilliary lemmas:

lemma *lemma6:*
"BirthdayBook (birthday, known) ⟹ birthday : NAME -|-> DATE"
by*(zstrip, zunfold BirthdayBook_def,simp add: Z2HOL)*

Notice that you actually *can* use the ' symbol in the lexis; However, this works in the standard HOL parser and has nothing to do with the Z stroke convention ...

lemma *lemma7:*
```
"⟦ BirthdayBook (birthday, {a. EX i: 1 .. hwm. names %^ i = a});
    BirthdayBook (birthday'a, {a. EX i: 1 .. hwm'. names' %^ i = a});
    BirthdayBook1 (dates, hwm, names); BirthdayBook1 (dates', hwm', names');
    AddBirthday1 (dateI, dates, dates', hwm, hwm', nameI, names, names');
    BirthdayBook (birthday', known');
    ALL i: 1 .. hwm. names %^ i ~= nameI;
    birthday' = insert (nameI, dateI) birthday;
    ALL i: 1 .. hwm. birthday %^ (names %^ i) = dates %^ i;
    ALL i: 1 .. hwm'. birthday'a %^ (names' %^ i) = dates' %^ i ⟧
  ⟹ dom birthday'a = insert nameI (dom birthday)"
```

apply*(subgoal_tac "nameI ~: dom birthday")*
apply*(simp add: insert_is_pfun)*
apply*(zunfold BirthdayBook_def,simp add: Z2HOL)*
apply*((erule conjE)+, drule sym, simp)*
apply*(zunfold AddBirthday1_def BirthdayBook1_def,simp add: maplet_def Z2HOL,clarify)*
defer *1*
apply*(zunfold BirthdayBook_def,simp add: maplet_def Z2HOL,clarify)*
apply*(drule_tac t="dom ?Z"* **in** *sym)+*
apply*(simp,blast)*

we reorient the two crucial equalities in the assumptions:

apply*(thin_tac "?X")*

```
apply(drule_tac t="dom ?Z" in sym)+
apply(rule set_ext, simp,safe,simp)
apply(case_tac "i=hwm+1",simp)
apply(rotate_tac -2)
apply(erule_tac x=i in ballE,simp)
```

... which yields a contradiction

```
apply(simp add: zpred_def)
apply(rule_tac x="hwm+1" in bexI,simp, simp,
      simp add: numb_range_def in_naturals[symmetric])
apply(rule_tac x=i in bexI)
apply(thin_tac "ALL x:?S. ?P x")+
apply(thin_tac "?T = ?U")+
apply(subst oplus_by_pair_apply2, simp add: numb_range_def,simp)
apply(simp add: numb_range_def)
done

lemma lemma8:
"⟦ BirthdayBook (birthday, {a. EX i: 1 .. hwm. names %^ i = a});
    BirthdayBook (birthday'a, {a. EX i: 1 .. hwm'. names' %^ i = a});
    BirthdayBook1 (dates, hwm, names); BirthdayBook1 (dates', hwm', names');
    AddBirthday1 (dateI, dates, dates', hwm, hwm', nameI, names, names');
    BirthdayBook (birthday', known');
    ALL i: 1 .. hwm. names %^ i ~= nameI;
    birthday' = insert (nameI, dateI) birthday;
    ALL i: 1 .. hwm. birthday %^ (names %^ i) = dates %^ i;
    ALL i: 1 .. hwm'. birthday'a %^ (names' %^ i) = dates' %^ i;
    i : dom birthday'a; dom birthday'a = dom (insert (nameI, dateI) birthday) ⟧
 ⟹ birthday'a %^ i = insert (nameI, dateI) birthday %^ i"
apply(subgoal_tac "nameI ~: dom birthday")
apply(simp add: insert_is_pfun)
apply(zunfold BirthdayBook_def,simp add: Z2HOL)
apply((erule conjE)+, drule sym, simp)
apply(zunfold AddBirthday1_def,zunfold BirthdayBook1_def,simp add: maplet_def Z2HOL,clarify)
defer 1
apply(zunfold BirthdayBook_def,simp add: maplet_def Z2HOL,clarify)
apply(drule_tac t="dom ?Z" in sym)+
apply(simp,blast)
apply(erule disjE, simp, (thin_tac "?T<:?S = ?U")+)
```

Case A: i is equal to $hwm + 1$. Tis boils down to:

```
 1. ⟦∀i∈1 .. hwm + 1. ∀j∈1 .. hwm + 1. i ≠ j ⟶ (names ⊕ {(hwm + 1, nameI)})(.i.)
≠ (names ⊕ {(hwm + 1, nameI)})(.j.); ∀i∈1 .. hwm. names(.i.) ≠ nameI; ∀i∈1 .. hwm. birthday(.(names(.i.)).)
= dates(.i.); ∀i∈1 .. hwm + 1. birthday'a(.((names ⊕ {(hwm + 1, nameI)})(.i.)).) = (dates
⊕ {(hwm + 1, dateI)})(.i.); dom birthday'a = insert nameI (dom birthday); nameI ∉ dom
birthday; birthday ∈ NAME ⇸ DATE; {a. ∃i∈1 .. hwm. names(.i.) = a} = dom birthday; birthday'a
∈ NAME ⇸ DATE; {a. ∃i∈1 .. hwm + 1. (names ⊕ {(hwm + 1, nameI)})(.i.) = a} = insert
nameI (dom birthday); names ∈ ℕ → NAME; ∀i∈1 .. hwm. nameI ≠ names(.i.); dates ∈ ℕ
→ DATE; hwm ∈ ℕ; ∀i∈1 .. hwm. ∀j∈1 .. hwm. i ≠ j ⟶ names(.i.) ≠ names(.j.); i = nameI⟧
⟹ birthday'a(.nameI.) = dateI
```

```
apply(drule_tac x="nameI" in eqset_imp_iff)
apply(simp,safe, thin_tac "?X")
```

apply(*erule_tac x="hwm+1"* **and** *A="1..hwm +1"* **in** *ballE,simp)*
apply(*simp add: numb_range_def in_naturals[symmetric]*)

Case B: i is less to $hwm + 1$, i.e. in the domain of *birthdaybook*:

1. ⋀y. ⟦∀ i∈1 .. hwm + 1. ∀ j∈1 .. hwm + 1. i ≠ j ⟶ (names ⊕ {(hwm + 1, nameI)})(.i.) ≠ (names ⊕ {(hwm + 1, nameI)})(.j.); ∀ i∈1 .. hwm. names(.i.) ≠ nameI; ∀ i∈1 .. hwm. birthday(.(n = dates(.i.); ∀ i∈1 .. hwm + 1. birthday'a(.((names ⊕ {(hwm + 1, nameI)})(.i.)).) = (dates ⊕ {(hwm + 1, dateI)})(.i.); dom birthday'a = insert nameI (dom birthday); nameI ∉ dom birthday; NAME ◁ birthday = birthday; birthday ∈ NAME ⇸ DATE; {a. ∃ i∈1 .. hwm. names(.i.) = a} = dom birthday; birthday'a ∈ NAME ⇸ DATE; {a. ∃ i∈1 .. hwm + 1. (names ⊕ {(hwm + 1, nameI)})(.i.) = a} = insert nameI (dom birthday); names ∈ ℕ → NAME; ∀ i∈1 .. hwm. nameI ≠ names(.i.); dates ∈ ℕ → DATE; hwm ∈ ℕ; ∀ i∈1 .. hwm. ∀ j∈1 .. hwm. i ≠ j ⟶ names(.i.) ≠ names(.j.); (i, y) ∈ birthday⟧ ⟹ birthday'a(.i.) = insert (nameI, dateI) birthday(.i.)

apply(*thin_tac "?X", (thin_tac "?T<:?S = ?U")+*)
apply(*drule_tac x=i* **in** *eqset_imp_iff*) **back**
apply(*simp, safe*)
apply(*rule_tac A="1 .. hwm+1"* **and** *x = i* **in** *ballE*)
apply *assumption*
apply(*subgoal_tac "((names (+) {(hwm + 1, nameI)}) %^ i) = names %^ i", simp*)
apply(*subst oplus_by_pair_apply2*)
apply(*simp add: numb_range_def in_naturals[symmetric]*)
apply(*rotate_tac 1*)
apply(*erule_tac x=i* **in** *ballE,simp,simp*)
apply(*subst oplus_by_pair_apply2*)

The proof conclusion is somewhat messy. The proof state is so cluttered up with facts, that it takes the automated procedures quite some time wo find the contradiction here. Thinning the assumption lists therefore greatly improves the proof speed.

apply(*(thin_tac "ALL x:?S. ?P x")+, (thin_tac "?T = ?U")+,(thin_tac "?X : ?Y ---> ?Z")+,
    simp add: numb_range_def in_naturals[symmetric], simp*)
apply(*(thin_tac "ALL x:?S. ?P x")+, (thin_tac "?T = ?U")+,(thin_tac "?X : ?Y ---> ?Z")+,
    thin_tac "?X",thin_tac "?X",thin_tac "?X",thin_tac "?X", thin_tac "?X"*)
apply(*simp add: numb_range_def in_naturals[symmetric]*)
done

po *Fun_Refinement.fwRefinementOp_AddBirthday_2*

To show:

1. ∀ asSet BirthdayBook • (SB ''birthday'' ~~> birthday, ''known'' ~~> known. ∀ asSet BirthdayBook • (SB ''birthday‘'' ~~> birthday‘, ''known‘'' ~~> known‘. ∀ asSet BirthdayBook1 • (SB ''dates'' ~~> dates, ''hwm'' ~~> hwm, ''names'' ~~> names. ∀ asSet BirthdayBook1 • (SB ''dates‘'' ~~> dates‘, ''hwm‘'' ~~> hwm‘, ''names‘'' ~~> names‘. ∀ date? name?. pre (SB ''birthday‘'' ~~> birthday‘, ''known‘'' ~~> known‘. SNAME AddBirthday (birthday, birthday‘, date?, known, known‘, name?)) ∧ SNAME Abs (birthday, dates, hwm, known, names) ∧ SNAME AddBirthday1 (date?, dates, dates‘, hwm, hwm‘, name?, names, names‘) ∧ SNAME Abs (birthday‘, dates‘, hwm‘, known‘, names‘) ⟶ SNAME AddBirthday (birthday, birthday‘, date?, known, known‘, name?)))))

After structural simplification:

apply(*zstrip, clarify, zelim_pre*)

apply(zunfold Abs_def AddBirthday_def,
    simp_all add: Z2HOL rel_appl_norm maplet_def,clarify)

This leads to the proof-state presented in Spivey proof (pp. 141):

1. ⋀birthday known birthday_96a known_96a dates hwm names dates_96 hwm_96 names_96 date? name? birthday_96 known_96. ⟦BirthdayBook (birthday_96a, {a. ∃ i∈1 .. hwm_96. names_96(.i.) = a}); BirthdayBook1 (dates, hwm, names); BirthdayBook1 (dates_96, hwm_96, names_96); AddBirthday1 (date?, dates, dates_96, hwm, hwm_96, name?, names, names_96); ∀ i∈1 .. hwm. birthday(.(names(.i.)).) = dates(.i.); ∀ i∈1 .. hwm_96. birthday_96a(.(names_96(.i.)).) = dates_96(.i.); BirthdayBook (birthday, {a. ∃ i∈1 .. hwm. names(.i.) = a}); BirthdayBook (insert (name?, date?) birthday, known_96); ∀ i∈1 .. hwm. names(.i.) ≠ name?⟧ ⟹ birthday_96a = insert (name?, date?) birthday

Now we apply, as suggested in pp. 14, the extensionality rule on partial functions:

apply(rule pfun_ext)

Extensionality works only under condition that both sides are in fact partial functions. Spiveys proof misses this detail.

apply(erule lemma6)+

Now we reach the mail case distinction shown on page 14: we have to show that both domains are equal, and that they are pointwise equal:

We first treat the domain equality: 1. ⋀birthday known birthday_96a known_96a dates hwm names dates_96 hwm_96 names_96 date? name? birthday_96 known_96. ⟦BirthdayBook (birthday_96a, {a. ∃ i∈1 .. hwm_96. names_96(.i.) = a}); BirthdayBook1 (dates, hwm, names); BirthdayBook1 (dates_96, hwm_96, names_96); AddBirthday1 (date?, dates, dates_96, hwm, hwm_96, name?, names, names_96); ∀ i∈1 .. hwm. birthday(.(names(.i.)).) = dates(.i.); ∀ i∈1 .. hwm_96. birthday_96a(.(names_96 = dates_96(.i.); BirthdayBook (birthday, {a. ∃ i∈1 .. hwm. names(.i.) = a}); BirthdayBook (insert (name?, date?) birthday, known_96); ∀ i∈1 .. hwm. names(.i.) ≠ name?⟧ ⟹ dom birthday_96a = dom (insert (name?, date?) birthday)

apply(simp (no_asm))
apply(rule_tac birthday'="insert (name?, date?) birthday" **in** lemma7,simp_all)

Now comes the part with the pointwise argument: 1. ⋀birthday birthday_96a dates hwm names dates_96 hwm_96 names_96 date? name? known_96 i. ⟦BirthdayBook (birthday_96a, {a. ∃ i∈1 .. hwm_96. names_96(.i.) = a}); BirthdayBook1 (dates, hwm, names); BirthdayBook1 (dates_96, hwm_96, names_96); AddBirthday1 (date?, dates, dates_96, hwm, hwm_96, name?, names, names_96); ∀ i∈1 .. hwm. birthday(.(names(.i.)).) = dates(.i.); ∀ i∈1 .. hwm_96. birthday_96a(.(names_96 = dates_96(.i.); BirthdayBook (birthday, {a. ∃ i∈1 .. hwm. names(.i.) = a}); BirthdayBook (insert (name?, date?) birthday, known_96); ∀ i∈1 .. hwm. names(.i.) ≠ name?; i = name? ∨ i ∈ dom birthday; dom birthday_96a = insert name? (dom birthday)⟧ ⟹ birthday_96a(.i.) = insert (name?, date?) birthday(.i.)

apply(rule_tac birthday'a="birthday_96a" **and** birthday'="insert (name?, date?) birthday"

    in lemma8,simp_all)
discharged

3.6 Global Checks

Now we check that all refinement conditions have indeed been proven:

check_po except ccOp ccState

This concludes the refinement proof based on Forward Simulation in the style of Spivey for the Birthdaybook example.

end

4 Include both Refinement Proofs

theory *BB*
imports *Fun_Refinement*

begin

end