Automatic Data Refinement

Peter Lammich

March 19, 2025

Abstract

We present the Autoref tool for Isabelle/HOL, which automatically refines algorithms specified over abstract concepts like maps and sets to algorithms over concrete implementations like red-black-trees, and produces a refinement theorem. It is based on ideas borrowed from relational parametricity due to Reynolds and Wadler. The tool allows for rapid prototyping of verified, executable algorithms. Moreover, it can be configured to fine-tune the result to the user's needs. Our tool is able to automatically instantiate generic algorithms, which greatly simplifies the implementation of executable data structures.

This AFP-entry provides the basic tool, which is then used by the Refinement and Collection Framework to provide automatic data refinement for the nondeterminism monad and various collection datastructures.

Contents

Chapter 1

Parametricity Solver

1.1 Relators

theory *Relators*
imports *../Lib/Refine-Lib*
begin

We define the concept of relators. The relation between a concrete type and an abstract type is expressed by a relation of type $('c \times 'a)$ *set*. For each composed type, say *'a list*, we can define a *relator*, that takes as argument a relation for the element type, and returns a relation for the list type. For most datatypes, there exists a *natural relator*. For algebraic datatypes, this is the relator that preserves the structure of the datatype, and changes the components. For example, *list-rel*::$('c \times 'a)$ *set* $\Rightarrow$ $('c$ *list* $\times$ $'a$ *list*$)$ *set* is the natural relator for lists.

However, relators can also be used to change the representation, and thus relate an implementation with an abstract type. For example, the relator *list-set-rel*::$('c \times 'a)$ *set* $\Rightarrow$ $('c$ *list* $\times$ $'a$ *set*$)$ *set* relates lists with the set of their elements.

In this theory, we define some basic notions for relators, and then define natural relators for all HOL-types, including the function type. For each relator, we also show a single-valuedness property, and initialize a solver for single-valued properties.

1.1.1 Basic Definitions

For smoother handling of relator unification, we require relator arguments to be applied by a special operator, such that we avoid higher-order unification problems. We try to set up some syntax to make this more transparent, and give relators a type-like prefix-syntax.

definition *relAPP*
:: $(('c1 \times 'a1)$ *set* $\Rightarrow$ -$)$ $\Rightarrow$ $('c1 \times 'a1)$ *set* $\Rightarrow$ -

where *relAPP f x* $\equiv$ *f x*

syntax *-rel-APP* :: *args* $\Rightarrow$ *'a* $\Rightarrow$ *'b* (‹⟨-⟩-› [0,900] 900)

syntax-consts *-rel-APP* == *relAPP*

translations
⟨x,xs⟩R == ⟨xs⟩(CONST relAPP R x)
⟨x⟩R == CONST relAPP R x

⟨ML⟩

1.1.2 Basic HOL Relators

Function

definition *fun-rel* **where**
fun-rel-def-internal: $fun\text{-}rel\ A\ B \equiv \{\ (f,f').\ \forall (a,a')\in A.\ (f\ a,\ f'\ a')\in B\ \}$
abbreviation *fun-rel-syn* (**infixr** ‹→› 60) **where** $A \rightarrow B \equiv \langle A,B\rangle fun\text{-}rel$

lemma *fun-rel-def*[*refine-rel-defs*]:
$A \rightarrow B \equiv \{\ (f,f').\ \forall (a,a')\in A.\ (f\ a,\ f'\ a')\in B\ \}$
⟨*proof*⟩

lemma *fun-relI*[*intro!*]: $[\![\bigwedge a\ a'.\ (a,a')\in A \Longrightarrow (f\ a,f'\ a')\in B]\!] \Longrightarrow (f,f')\in A\rightarrow B$
⟨*proof*⟩

lemma *fun-relD*:
shows $((f,f')\in(A\rightarrow B)) \Longrightarrow$
$(\bigwedge x\ x'.\ [\![\ (x,x')\in A\ ]\!] \Longrightarrow (f\ x,\ f'\ x')\in B)$
⟨*proof*⟩

lemma *fun-relD1*:
assumes $(f,f')\in Ra\rightarrow Rr$
assumes $f\ x = r$
shows $\forall x'.\ (x,x')\in Ra \longrightarrow (r,f'\ x')\in Rr$
⟨*proof*⟩

lemma *fun-relD2*:
assumes $(f,f')\in Ra\rightarrow Rr$
assumes $f'\ x' = r'$
shows $\forall x.\ (x,x')\in Ra \longrightarrow (f\ x,r')\in Rr$
⟨*proof*⟩

lemma *fun-relE1*:
assumes $(f,f')\in Id \rightarrow Rv$
assumes $t' = f'\ x$
shows $(f\ x,t')\in Rv$ ⟨*proof*⟩

lemma *fun-relE2*:
  assumes $(f,f')\in Id \rightarrow Rv$
  assumes $t = f\ x$
  shows $(t,f'\ x)\in Rv$ ⟨*proof*⟩

Terminal Types

abbreviation *unit-rel* :: (*unit*×*unit*) *set* **where** *unit-rel* == *Id*

abbreviation *nat-rel* ≡ *Id*::(*nat*×-) *set*
abbreviation *int-rel* ≡ *Id*::(*int*×-) *set*
abbreviation *bool-rel* ≡ *Id*::(*bool*×-) *set*

Product

definition *prod-rel* **where**
  prod-rel-def-internal: *prod-rel R1 R2*
    $\equiv \{ ((a,b),(a',b'))\ .\ (a,a')\in R1 \wedge (b,b')\in R2 \}$

abbreviation *prod-rel-syn* (**infixr** ‹$\times_r$› *70*) **where** $a\times_r b \equiv \langle a,b\rangle$*prod-rel*

lemma *prod-rel-def*[*refine-rel-defs*]:
  $(\langle R1,R2\rangle prod\text{-}rel) \equiv \{ ((a,b),(a',b'))\ .\ (a,a')\in R1 \wedge (b,b')\in R2 \}$
  ⟨*proof*⟩

lemma *prod-relI*: $[\![(a,a')\in R1;\ (b,b')\in R2]\!] \Longrightarrow ((a,b),(a',b'))\in\langle R1,R2\rangle prod\text{-}rel$
  ⟨*proof*⟩
lemma *prod-relE*:
  assumes $(p,p')\in\langle R1,R2\rangle prod\text{-}rel$
  obtains $a\ b\ a'\ b'$ **where** $p=(a,b)$ **and** $p'=(a',b')$
  and $(a,a')\in R1$ **and** $(b,b')\in R2$
  ⟨*proof*⟩

lemma *prod-rel-simp*[*simp*]:
  $((a,b),(a',b'))\in\langle R1,R2\rangle prod\text{-}rel \longleftrightarrow (a,a')\in R1 \wedge (b,b')\in R2$
  ⟨*proof*⟩

lemma *in-Domain-prod-rel-iff*[*iff*]: $(a,b)\in Domain\ (A\times_r B) \longleftrightarrow a\in Domain\ A \wedge b\in Domain\ B$
  ⟨*proof*⟩

lemma *prod-rel-comp*: $(A \times_r B)\ O\ (C \times_r D) = (A\ O\ C) \times_r (B\ O\ D)$
  ⟨*proof*⟩

Option

definition *option-rel* **where**
  option-rel-def-internal:
  option-rel $R \equiv \{ (Some\ a,Some\ a') \mid a\ a'.\ (a,a')\in R \} \cup \{(None,None)\}$

lemma *option-rel-def*[*refine-rel-defs*]:
$\langle R\rangle$*option-rel* $\equiv$ { (*Some a*,*Some a'*) | *a a'*. (*a*,*a'*)$\in$*R* } $\cup$ {(*None*,*None*)}
$\langle$*proof*$\rangle$

lemma *option-relI*:
(*None*,*None*)$\in\langle R\rangle$ *option-rel*
$[\![$ (*a*,*a'*)$\in$*R* $]\!] \Longrightarrow$ (*Some a*, *Some a'*)$\in\langle R\rangle$*option-rel*
$\langle$*proof*$\rangle$

lemma *option-relE*:
assumes (*x*,*x'*)$\in\langle R\rangle$*option-rel*
obtains *x*=*None* **and** *x'*=*None*
| *a a'* **where** *x*=*Some a* **and** *x'*=*Some a'* **and** (*a*,*a'*)$\in$*R*
$\langle$*proof*$\rangle$

lemma *option-rel-simp*[*simp*]:
(*None*,*a*)$\in\langle R\rangle$*option-rel* $\longleftrightarrow$ *a*=*None*
(*c*,*None*)$\in\langle R\rangle$*option-rel* $\longleftrightarrow$ *c*=*None*
(*Some x*,*Some y*)$\in\langle R\rangle$*option-rel* $\longleftrightarrow$ (*x*,*y*)$\in$*R*
$\langle$*proof*$\rangle$

Sum

definition *sum-rel* **where** *sum-rel-def-internal*:
sum-rel Rl Rr
$\equiv$ { (*Inl a*, *Inl a'*) | *a a'*. (*a*,*a'*)$\in$*Rl* } $\cup$
{ (*Inr a*, *Inr a'*) | *a a'*. (*a*,*a'*)$\in$*Rr* }

lemma *sum-rel-def*[*refine-rel-defs*]:
$\langle$*Rl*,*Rr*$\rangle$*sum-rel* $\equiv$
{ (*Inl a*, *Inl a'*) | *a a'*. (*a*,*a'*)$\in$*Rl* } $\cup$
{ (*Inr a*, *Inr a'*) | *a a'*. (*a*,*a'*)$\in$*Rr* }
$\langle$*proof*$\rangle$

lemma *sum-rel-simp*[*simp*]:
$\bigwedge$*a a'*. (*Inl a*, *Inl a'*) $\in \langle$*Rl*,*Rr*$\rangle$*sum-rel* $\longleftrightarrow$ (*a*,*a'*)$\in$*Rl*
$\bigwedge$*a a'*. (*Inr a*, *Inr a'*) $\in \langle$*Rl*,*Rr*$\rangle$*sum-rel* $\longleftrightarrow$ (*a*,*a'*)$\in$*Rr*
$\bigwedge$*a a'*. (*Inl a*, *Inr a'*) $\notin \langle$*Rl*,*Rr*$\rangle$*sum-rel*
$\bigwedge$*a a'*. (*Inr a*, *Inl a'*) $\notin \langle$*Rl*,*Rr*$\rangle$*sum-rel*
$\langle$*proof*$\rangle$

lemma *sum-relI*:
(*l*,*l'*)$\in$*Rl* $\Longrightarrow$ (*Inl l*, *Inl l'*) $\in \langle$*Rl*,*Rr*$\rangle$*sum-rel*
(*r*,*r'*)$\in$*Rr* $\Longrightarrow$ (*Inr r*, *Inr r'*) $\in \langle$*Rl*,*Rr*$\rangle$*sum-rel*
$\langle$*proof*$\rangle$

lemma *sum-relE*:
assumes (*x*,*x'*)$\in\langle$*Rl*,*Rr*$\rangle$*sum-rel*
obtains

l l' **where** x=*Inl* l **and** x'=*Inl* l' **and** $(l,l')\in Rl$
| r r' **where** x=*Inr* r **and** x'=*Inr* r' **and** $(r,r')\in Rr$
⟨*proof*⟩

Lists

definition *list-rel* **where** *list-rel-def-internal*:
list-rel $R \equiv \{(l,l').$ *list-all2* $(\lambda x\ x'.\ (x,x')\in R)\ l\ l'\}$

lemma *list-rel-def*[*refine-rel-defs*]:
$\langle R\rangle$*list-rel* $\equiv \{(l,l').$ *list-all2* $(\lambda x\ x'.\ (x,x')\in R)\ l\ l'\}$
⟨*proof*⟩

lemma *list-rel-induct*[*induct set*,*consumes 1*, *case-names Nil Cons*]:
assumes $(l,l')\in\langle R\rangle$ *list-rel*
assumes P [] []
assumes $\bigwedge x\ x'\ l\ l'.$ $[\![$ $(x,x')\in R$; $(l,l')\in\langle R\rangle$*list-rel*; $P\ l\ l'$ $]\!]$
$\Longrightarrow P\ (x\#l)\ (x'\#l')$
shows $P\ l\ l'$
⟨*proof*⟩

lemma *list-rel-eq-listrel*: *list-rel* = *listrel*
⟨*proof*⟩

lemma *list-relI*:
$([],[])\in\langle R\rangle$*list-rel*
$[\![$ $(x,x')\in R$; $(l,l')\in\langle R\rangle$*list-rel* $]\!]$ $\Longrightarrow$ $(x\#l,x'\#l')\in\langle R\rangle$*list-rel*
⟨*proof*⟩

lemma *list-rel-simp*[*simp*]:
$([],l')\in\langle R\rangle$*list-rel* $\longleftrightarrow$ l'=[]
$(l,[])\in\langle R\rangle$*list-rel* $\longleftrightarrow$ l=[]
$([],[])\in\langle R\rangle$*list-rel*
$(x\#l,x'\#l')\in\langle R\rangle$*list-rel* $\longleftrightarrow$ $(x,x')\in R \wedge (l,l')\in\langle R\rangle$*list-rel*
⟨*proof*⟩

lemma *list-relE1*:
assumes $(l,[])\in\langle R\rangle$*list-rel* **obtains** l=[] ⟨*proof*⟩

lemma *list-relE2*:
assumes $([],l)\in\langle R\rangle$*list-rel* **obtains** l=[] ⟨*proof*⟩

lemma *list-relE3*:
assumes $(x\#xs,l')\in\langle R\rangle$*list-rel* **obtains** x' xs' **where**
$l'=x'\#xs'$ **and** $(x,x')\in R$ **and** $(xs,xs')\in\langle R\rangle$*list-rel*
⟨*proof*⟩

lemma *list-relE4*:
assumes $(l,x'\#xs')\in\langle R\rangle$*list-rel* **obtains** x xs **where**

l=*x*#*xs* **and** (*x*,*x'*)∈*R* **and** (*xs*,*xs'*)∈⟨*R*⟩*list-rel*
⟨*proof*⟩

lemmas *list-relE* = *list-relE1* *list-relE2* *list-relE3* *list-relE4*

lemma *list-rel-imp-same-length*:
$(l,\ l') \in \langle R\rangle \mathit{list\text{-}rel} \Longrightarrow \mathit{length}\ l = \mathit{length}\ l'$
⟨*proof*⟩

lemma *list-rel-split-right-iff*:
$(x\#xs,l)\in\langle R\rangle \mathit{list\text{-}rel} \longleftrightarrow (\exists\, y\ ys.\ l=y\#ys \wedge (x,y)\in R \wedge (xs,ys)\in\langle R\rangle \mathit{list\text{-}rel})$
⟨*proof*⟩
lemma *list-rel-split-left-iff*:
$(l,y\#ys)\in\langle R\rangle \mathit{list\text{-}rel} \longleftrightarrow (\exists\, x\ xs.\ l=x\#xs \wedge (x,y)\in R \wedge (xs,ys)\in\langle R\rangle \mathit{list\text{-}rel})$
⟨*proof*⟩

Sets

Pointwise refinement: The abstract set is the image of the concrete set, and the concrete set only contains elements that have an abstract counterpart

definition *set-rel* **where**
set-rel-def-internal:
$\mathit{set\text{-}rel}\ R \equiv \{(A,B).\ (\forall\, x\in A.\ \exists\, y\in B.\ (x,y)\in R) \wedge (\forall\, y\in B.\ \exists\, x\in A.\ (x,y)\in R)\}$

term *set-rel*

lemma *set-rel-def*[*refine-rel-defs*]:
$\langle R\rangle \mathit{set\text{-}rel} \equiv \{(A,B).\ (\forall\, x\in A.\ \exists\, y\in B.\ (x,y)\in R) \wedge (\forall\, y\in B.\ \exists\, x\in A.\ (x,y)\in R)\}$
⟨*proof*⟩

lemma *set-rel-alt*: $\langle R\rangle \mathit{set\text{-}rel} = \{(A,B).\ A \subseteq R^{-1}``B \wedge B \subseteq R``A\}$
⟨*proof*⟩

lemma *set-relI*[*intro?*]:
assumes $\bigwedge x.\ x\in A \Longrightarrow \exists\, y\in B.\ (x,y)\in R$
assumes $\bigwedge y.\ y\in B \Longrightarrow \exists\, x\in A.\ (x,y)\in R$
shows (*A*,*B*)∈⟨*R*⟩*set-rel*
⟨*proof*⟩

Original definition of *set-rel* in refinement framework. Abandoned in favour of more symmetric definition above:

definition *old-set-rel* **where** *old-set-rel-def-internal*:
$\mathit{old\text{-}set\text{-}rel}\ R \equiv \{(S,S').\ S'=R``S \wedge S\subseteq \mathit{Domain}\ R\}$

lemma *old-set-rel-def*[*refine-rel-defs*]:
$\langle R\rangle \mathit{old\text{-}set\text{-}rel} \equiv \{(S,S').\ S'=R``S \wedge S\subseteq \mathit{Domain}\ R\}$

⟨*proof*⟩

Old definition coincides with new definition for single-valued element relations. This is probably the reason why the old definition worked for most applications.

lemma *old-set-rel-sv-eq*: $single\text{-}valued\ R \Longrightarrow \langle R\rangle old\text{-}set\text{-}rel = \langle R\rangle set\text{-}rel$
⟨*proof*⟩

lemma *set-rel-simp*[*simp*]:
$(\{\},\{\}) \in \langle R\rangle set\text{-}rel$
⟨*proof*⟩

lemma *set-rel-empty-iff*[*simp*]:
$(\{\},y) \in \langle A\rangle set\text{-}rel \longleftrightarrow y=\{\}$
$(x,\{\}) \in \langle A\rangle set\text{-}rel \longleftrightarrow x=\{\}$
⟨*proof*⟩

lemma *set-relD1*: $(s,s') \in \langle R\rangle set\text{-}rel \Longrightarrow x \in s \Longrightarrow \exists\, x' \in s'.\ (x,x') \in R$
⟨*proof*⟩

lemma *set-relD2*: $(s,s') \in \langle R\rangle set\text{-}rel \Longrightarrow x' \in s' \Longrightarrow \exists\, x \in s.\ (x,x') \in R$
⟨*proof*⟩

lemma *set-relE1*[*consumes 2*]:
assumes $(s,s') \in \langle R\rangle set\text{-}rel \quad x \in s$
obtains x' **where** $x' \in s' \quad (x,x') \in R$
⟨*proof*⟩

lemma *set-relE2*[*consumes 2*]:
assumes $(s,s') \in \langle R\rangle set\text{-}rel \quad x' \in s'$
obtains x **where** $x \in s \quad (x,x') \in R$
⟨*proof*⟩

1.1.3 Automation

A solver for relator properties

lemma *relprop-triggers*:
$\bigwedge R.\ single\text{-}valued\ R \Longrightarrow single\text{-}valued\ R$
$\bigwedge R.\ R=Id \Longrightarrow R=Id$
$\bigwedge R.\ R=Id \Longrightarrow Id=R$
$\bigwedge R.\ Range\ R = UNIV \Longrightarrow Range\ R = UNIV$
$\bigwedge R.\ Range\ R = UNIV \Longrightarrow UNIV = Range\ R$
$\bigwedge R\ R'.\ R \subseteq R' \Longrightarrow R \subseteq R'$
⟨*proof*⟩

⟨*ML*⟩

lemma
relprop-id-orient[*relator-props*]: $R=Id \Longrightarrow Id=R$ **and**
relprop-eq-refl[*solve-relator-props*]: $t = t$
⟨*proof*⟩

lemma
relprop-UNIV-orient[*relator-props*]: $R=UNIV \Longrightarrow UNIV=R$
⟨*proof*⟩

ML-Level utilities

⟨*ML*⟩

1.1.4 Setup

Natural Relators

declare [[*natural-relator*
unit-rel int-rel nat-rel bool-rel
fun-rel prod-rel option-rel sum-rel list-rel
]]

⟨*ML*⟩

Additional Properties

lemmas [*relator-props*] =
single-valued-Id
subset-refl
refl

lemma *eq-UNIV-iff*: $S=UNIV \longleftrightarrow (\forall x.\ x \in S)$ ⟨*proof*⟩

lemma *fun-rel-sv*[*relator-props*]:
assumes *RAN*: *Range Ra* = *UNIV*
assumes *SV*: *single-valued Rv*
shows *single-valued* $(Ra \to Rv)$
⟨*proof*⟩

lemmas [*relator-props*] = *Range-Id*

lemma *fun-rel-id*[*relator-props*]: $[\![R1=Id;\ R2=Id]\!] \Longrightarrow R1 \to R2 = Id$
⟨*proof*⟩

lemma *fun-rel-id-simp*[*simp*]: $Id \to Id = Id$ ⟨*proof*⟩

lemma *fun-rel-comp-dist*[*relator-props*]:
$(R1 \rightarrow R2)\ O\ (R3 \rightarrow R4) \subseteq ((R1\ O\ R3) \rightarrow (R2\ O\ R4))$
⟨*proof*⟩

lemma *fun-rel-mono*[*relator-props*]: $[\![\ R1 \subseteq R2;\ R3 \subseteq R4\ ]\!] \Longrightarrow R2 \rightarrow R3 \subseteq R1 \rightarrow R4$
⟨*proof*⟩

lemma *prod-rel-sv*[*relator-props*]:
$[\![$*single-valued R1*; *single-valued R2*$]\!] \Longrightarrow$ *single-valued* (⟨*R1*,*R2*⟩*prod-rel*)
⟨*proof*⟩

lemma *prod-rel-id*[*relator-props*]: $[\![R1=Id;\ R2=Id]\!] \Longrightarrow$ ⟨*R1*,*R2*⟩*prod-rel* = *Id*
⟨*proof*⟩

lemma *prod-rel-id-simp*[*simp*]: ⟨*Id*,*Id*⟩*prod-rel* = *Id* ⟨*proof*⟩

lemma *prod-rel-mono*[*relator-props*]:
$[\![\ R2 \subseteq R1;\ R3 \subseteq R4\ ]\!] \Longrightarrow$ ⟨*R2*,*R3*⟩*prod-rel* ⊆ ⟨*R1*,*R4*⟩*prod-rel*
⟨*proof*⟩

lemma *prod-rel-range*[*relator-props*]: $[\![$*Range Ra*=*UNIV*; *Range Rb*=*UNIV*$]\!]$
$\Longrightarrow$ *Range* (⟨*Ra*,*Rb*⟩*prod-rel*) = *UNIV*
⟨*proof*⟩

lemma *option-rel-sv*[*relator-props*]:
$[\![$*single-valued R*$]\!] \Longrightarrow$ *single-valued* (⟨*R*⟩*option-rel*)
⟨*proof*⟩

lemma *option-rel-id*[*relator-props*]:
R=*Id* $\Longrightarrow$ ⟨*R*⟩*option-rel* = *Id* ⟨*proof*⟩

lemma *option-rel-id-simp*[*simp*]: ⟨*Id*⟩*option-rel* = *Id* ⟨*proof*⟩

lemma *option-rel-mono*[*relator-props*]: $R \subseteq R' \Longrightarrow$ ⟨*R*⟩*option-rel* ⊆ ⟨*R'*⟩*option-rel*
⟨*proof*⟩

lemma *option-rel-range*: *Range R* = *UNIV* $\Longrightarrow$ *Range* (⟨*R*⟩*option-rel*) = *UNIV*
⟨*proof*⟩

lemma *option-rel-inter*[*simp*]: ⟨*R1* ∩ *R2*⟩*option-rel* = ⟨*R1*⟩*option-rel* ∩ ⟨*R2*⟩*option-rel*
⟨*proof*⟩

lemma *option-rel-constraint*[*simp*]:
(*x*,*x*)∈⟨*UNIV*×*C*⟩*option-rel* $\longleftrightarrow$ (∀ *v*. *x*=*Some v* $\longrightarrow$ *v*∈*C*)
⟨*proof*⟩

lemma *sum-rel-sv*[*relator-props*]:

$\llbracket$*single-valued Rl*; *single-valued Rr*$\rrbracket \Longrightarrow$ *single-valued* ($\langle Rl,Rr\rangle$*sum-rel*)
⟨*proof*⟩

lemma *sum-rel-id*[*relator-props*]: $\llbracket Rl=Id;\ Rr=Id\rrbracket \Longrightarrow \langle Rl,Rr\rangle sum\text{-}rel = Id$
⟨*proof*⟩

lemma *sum-rel-id-simp*[*simp*]: $\langle Id,Id\rangle sum\text{-}rel = Id$ ⟨*proof*⟩

lemma *sum-rel-mono*[*relator-props*]:
$\llbracket\ Rl\subseteq Rl';\ Rr\subseteq Rr'\ \rrbracket \Longrightarrow \langle Rl,Rr\rangle sum\text{-}rel \subseteq \langle Rl',Rr'\rangle sum\text{-}rel$
⟨*proof*⟩

lemma *sum-rel-range*[*relator-props*]:
$\llbracket$ *Range Rl=UNIV*; *Range Rr=UNIV* $\rrbracket \Longrightarrow$ *Range* ($\langle Rl,Rr\rangle$*sum-rel*) = *UNIV*
⟨*proof*⟩

lemma *list-rel-sv-iff*:
single-valued ($\langle R\rangle$*list-rel*) $\longleftrightarrow$ *single-valued R*
⟨*proof*⟩

lemma *list-rel-sv*[*relator-props*]:
single-valued R $\Longrightarrow$ *single-valued* ($\langle R\rangle$*list-rel*)
⟨*proof*⟩

lemma *list-rel-id*[*relator-props*]: $\llbracket R=Id\rrbracket \Longrightarrow \langle R\rangle list\text{-}rel = Id$
⟨*proof*⟩

lemma *list-rel-id-simp*[*simp*]: $\langle Id\rangle list\text{-}rel = Id$ ⟨*proof*⟩

lemma *list-rel-mono*[*relator-props*]:
assumes *A*: $R\subseteq R'$
shows $\langle R\rangle list\text{-}rel \subseteq \langle R'\rangle list\text{-}rel$
⟨*proof*⟩

lemma *list-rel-range*[*relator-props*]:
assumes *A*: *Range R* = *UNIV*
shows *Range* ($\langle R\rangle$*list-rel*) = *UNIV*
⟨*proof*⟩

lemma *bijective-imp-sv*:
bijective R $\Longrightarrow$ *single-valued R*
bijective R $\Longrightarrow$ *single-valued* (R^{-1})
⟨*proof*⟩

declare *bijective-Id*[*relator-props*]
declare *bijective-Empty*[*relator-props*]

Pointwise refinement for set types:

lemma *set-rel-sv*[*relator-props*]:
single-valued $R \Longrightarrow$ *single-valued* ($\langle R\rangle$*set-rel*)
$\langle$*proof*$\rangle$

lemma *set-rel-id*[*relator-props*]: R=*Id* $\Longrightarrow \langle R\rangle$*set-rel* = *Id*
$\langle$*proof*$\rangle$

lemma *set-rel-id-simp*[*simp*]: $\langle$*Id*$\rangle$*set-rel* = *Id* $\langle$*proof*$\rangle$

lemma *set-rel-csv*[*relator-props*]:
$[\![$ *single-valued* (R^{-1}) $]\!]$
$\Longrightarrow$ *single-valued* $((\langle R\rangle set\text{-}rel)^{-1})$
$\langle$*proof*$\rangle$

1.1.5 Invariant and Abstraction

Quite often, a relation can be described as combination of an abstraction function and an invariant, such that the invariant describes valid values on the concrete domain, and the abstraction function maps valid concrete values to its corresponding abstract value.

definition *build-rel* **where**
build-rel α $I \equiv \{(c,a)\ .\ a=\alpha\ c \wedge I\ c\}$
abbreviation *br*$\equiv$*build-rel*
lemmas *br-def*[*refine-rel-defs*] = *build-rel-def*

lemma *in-br-conv*: $(c,a)\in br\ \alpha\ I \longleftrightarrow a=\alpha\ c \wedge I\ c$
$\langle$*proof*$\rangle$

lemma *brI*[*intro?*]: $[\![\ a=\alpha\ c;\ I\ c\ ]\!] \Longrightarrow (c,a)\in br\ \alpha\ I$
$\langle$*proof*$\rangle$

lemma *br-id*[*simp*]: *br id* (λ-. *True*) = *Id*
$\langle$*proof*$\rangle$

lemma *br-chain*:
(*build-rel* β J) O (*build-rel* α I) = *build-rel* $(\alpha\circ\beta)$ $(\lambda s.\ J\ s \wedge I\ (\beta\ s))$
$\langle$*proof*$\rangle$

lemma *br-sv*[*simp*, *intro*!,*relator-props*]: *single-valued* (*br* α I)
$\langle$*proof*$\rangle$

lemma *converse-br-sv-iff*[*simp*]:
single-valued (*converse* (*br* α I)) $\longleftrightarrow$ *inj-on* α (*Collect* I)
$\langle$*proof*$\rangle$

lemmas [*relator-props*] = *single-valued-relcomp*

lemma *br-comp-alt*: *br* α I O R = $\{\ (c,a)\ .\ I\ c \wedge (\alpha\ c,a)\in R\ \}$

$\langle proof \rangle$

lemma *br-comp-alt'*:
$\{(c,a) \ . \ a=\alpha\ c \wedge I\ c\}\ O\ R = \{\ (c,a)\ .\ I\ c \wedge (\alpha\ c,a)\in R\ \}$
$\langle proof \rangle$

lemma *single-valued-as-brE*:
assumes *single-valued R*
obtains α *invar* **where** $R = br\ \alpha\ invar$
$\langle proof \rangle$

lemma *sv-add-invar*:
single-valued $R \Longrightarrow$ *single-valued* $\{(c,\ a).\ (c,\ a) \in R \wedge I\ c\}$
$\langle proof \rangle$

lemma *br-Image-conv*[*simp*]: $br\ \alpha\ I\ `` \ S = \{\alpha\ x \mid x.\ x\in S \wedge I\ x\}$
$\langle proof \rangle$

1.1.6 Miscellaneous

lemma *rel-cong*: $(f,g)\in Id \Longrightarrow (x,y)\in Id \Longrightarrow (f\ x,\ g\ y)\in Id$ $\langle proof \rangle$
lemma *rel-fun-cong*: $(f,g)\in Id \Longrightarrow (f\ x,\ g\ x)\in Id$ $\langle proof \rangle$
lemma *rel-arg-cong*: $(x,y)\in Id \Longrightarrow (f\ x,\ f\ y)\in Id$ $\langle proof \rangle$

1.1.7 Conversion between Predicate and Set Based Relators

Autoref uses set-based relators of type $('a \times 'b)$ *set*, while the transfer and lifting package of Isabelle/HOL uses predicate based relators of type $'a \Rightarrow 'b \Rightarrow$ *bool*. This section defines some utilities to convert between the two.

definition *rel2p* $R\ x\ y \equiv (x,y)\in R$
definition *p2rel* $P \equiv \{(x,y).\ P\ x\ y\}$

lemma *rel2pD*: $[\![rel2p\ R\ a\ b]\!] \Longrightarrow (a,b)\in R$ $\langle proof \rangle$
lemma *p2relD*: $[\![(a,b) \in p2rel\ R]\!] \Longrightarrow R\ a\ b$ $\langle proof \rangle$

lemma *rel2p-inv*[*simp*]:
rel2p (*p2rel* P) = P
p2rel (*rel2p* R) = R
$\langle proof \rangle$

named-theorems *rel2p*
named-theorems *p2rel*

lemma *rel2p-dflt*[*rel2p*]:
rel2p $Id = (=)$
rel2p $(A\rightarrow B)$ = *rel-fun* (*rel2p* A) (*rel2p* B)
rel2p $(A\times_r B)$ = *rel-prod* (*rel2p* A) (*rel2p* B)
rel2p $(\langle A,B\rangle$*sum-rel*) = *rel-sum* (*rel2p* A) (*rel2p* B)
rel2p $(\langle A\rangle$*option-rel*) = *rel-option* (*rel2p* A)

rel2p $(\langle A\rangle$*list-rel*$)$ = *list-all2* $($*rel2p* $A)$
$\langle$*proof*$\rangle$

lemma *p2rel-dflt*[*p2rel*]:
p2rel $(=)$ = *Id*
p2rel $($*rel-fun* A $B)$ = *p2rel* $A \rightarrow$ *p2rel* B
p2rel $($*rel-prod* A $B)$ = *p2rel* $A \times_r$ *p2rel* B
p2rel $($*rel-sum* A $B)$ = $\langle$*p2rel* A, *p2rel* $B\rangle$*sum-rel*
p2rel $($*rel-option* $A)$ = $\langle$*p2rel* $A\rangle$*option-rel*
p2rel $($*list-all2* $A)$ = $\langle$*p2rel* $A\rangle$*list-rel*
$\langle$*proof*$\rangle$

lemma [*rel2p*]: *rel2p* $(\langle A\rangle$*set-rel*$)$ = *rel-set* $($*rel2p* $A)$
$\langle$*proof*$\rangle$

lemma [*p2rel*]: *left-unique* $A \Longrightarrow$ *p2rel* $($*rel-set* $A)$ = $(\langle$*p2rel* $A\rangle$*set-rel*$)$
$\langle$*proof*$\rangle$

lemma *rel2p-comp*: *rel2p* A *OO* *rel2p* B = *rel2p* $(A\ O\ B)$
$\langle$*proof*$\rangle$

lemma *rel2p-inj*[*simp*]: *rel2p* A = *rel2p* $B \longleftrightarrow A{=}B$
$\langle$*proof*$\rangle$

lemma *rel2p-left-unique*: *left-unique* $($*rel2p* $A)$ = *single-valued* (A^{-1})
$\langle$*proof*$\rangle$

lemma *rel2p-right-unique*: *right-unique* $($*rel2p* $A)$ = *single-valued* A
$\langle$*proof*$\rangle$

lemma *rel2p-bi-unique*: *bi-unique* $($*rel2p* $A) \longleftrightarrow$ *single-valued* $A \wedge$ *single-valued* (A^{-1})
$\langle$*proof*$\rangle$

lemma *p2rel-left-unique*: *single-valued* $(($*p2rel* $A)^{-1})$ = *left-unique* A
$\langle$*proof*$\rangle$

lemma *p2rel-right-unique*: *single-valued* $($*p2rel* $A)$ = *right-unique* A
$\langle$*proof*$\rangle$

1.1.8 More Properties

lemma *list-rel-compp*: $\langle A\ O\ B\rangle$*list-rel* = $\langle A\rangle$*list-rel* O $\langle B\rangle$*list-rel*
$\langle$*proof*$\rangle$

lemma *option-rel-compp*: $\langle A\ O\ B\rangle$*option-rel* = $\langle A\rangle$*option-rel* O $\langle B\rangle$*option-rel*
$\langle$*proof*$\rangle$

lemma *prod-rel-compp*: $\langle A\ O\ B,\ C\ O\ D\rangle$*prod-rel* = $\langle A,C\rangle$*prod-rel* O $\langle B,D\rangle$*prod-rel*
⟨*proof*⟩

lemma *sum-rel-compp*: $\langle A\ O\ B,\ C\ O\ D\rangle$*sum-rel* = $\langle A,C\rangle$*sum-rel* O $\langle B,D\rangle$*sum-rel*
⟨*proof*⟩

lemma *set-rel-compp*: $\langle A\ O\ B\rangle$*set-rel* = $\langle A\rangle$*set-rel* O $\langle B\rangle$*set-rel*
⟨*proof*⟩

lemma *map-in-list-rel-conv*:
shows $(l,\ map\ \alpha\ l) \in \langle br\ \alpha\ I\rangle$*list-rel* $\longleftrightarrow (\forall x{\in}set\ l.\ I\ x)$
⟨*proof*⟩

lemma *br-set-rel-alt*: $(s',s){\in}\langle br\ \alpha\ I\rangle$*set-rel* $\longleftrightarrow (s{=}\alpha`s' \wedge (\forall x{\in}s'.\ I\ x))$
⟨*proof*⟩

lemma *finite-Image-sv*: *single-valued* $R \Longrightarrow$ *finite* $s \Longrightarrow$ *finite* $(R``s)$
⟨*proof*⟩

lemma *finite-set-rel-transfer*: $[\![(s,s'){\in}\langle R\rangle$*set-rel*; *single-valued* R; *finite* $s]\!] \Longrightarrow$ *finite* s'
⟨*proof*⟩

lemma *finite-set-rel-transfer-back*: $[\![(s,s'){\in}\langle R\rangle$*set-rel*; *single-valued* (R^{-1}); *finite* $s']\!] \Longrightarrow$ *finite* s
⟨*proof*⟩

end

1.2 Basic Parametricity Reasoning

theory *Param-Tool*
imports *Relators*
begin

1.2.1 Auxiliary Lemmas

lemma *tag-both*: $[\![\ (Let\ x\ f,Let\ x'\ f'){\in}R\ ]\!] \Longrightarrow (f\ x,f'\ x'){\in}R$ ⟨*proof*⟩
lemma *tag-rhs*: $[\![\ (c,Let\ x\ f){\in}R\ ]\!] \Longrightarrow (c,f\ x){\in}R$ ⟨*proof*⟩
lemma *tag-lhs*: $[\![\ (Let\ x\ f,a){\in}R\ ]\!] \Longrightarrow (f\ x,a){\in}R$ ⟨*proof*⟩

lemma *tagged-fun-relD-both*:
$[\![\ (f,f'){\in}A{\rightarrow}B;\ (x,x'){\in}A\ ]\!] \Longrightarrow (Let\ x\ f,Let\ x'\ f'){\in}B$
and *tagged-fun-relD-rhs*: $[\![\ (f,f'){\in}A{\rightarrow}B;\ (x,x'){\in}A\ ]\!] \Longrightarrow (f\ x,Let\ x'\ f'){\in}B$
and *tagged-fun-relD-lhs*: $[\![\ (f,f'){\in}A{\rightarrow}B;\ (x,x'){\in}A\ ]\!] \Longrightarrow (Let\ x\ f,f'\ x'){\in}B$
and *tagged-fun-relD-none*: $[\![\ (f,f'){\in}A{\rightarrow}B;\ (x,x'){\in}A\ ]\!] \Longrightarrow (f\ x,f'\ x'){\in}B$

⟨*proof*⟩

1.2.2 ML-Setup

⟨*ML*⟩

1.2.3 Convenience Tools

⟨*ML*⟩

end

1.3 Parametricity Theorems for HOL

theory *Param-HOL*
imports *Param-Tool*
begin

1.3.1 Sets

lemma *param-empty*[*param*]:
$(\{\},\{\}) \in \langle R\rangle \mathit{set\text{-}rel}$ ⟨*proof*⟩

lemma *param-member*[*param*]:
$[\![\mathit{single\text{-}valued}\ R;\ \mathit{single\text{-}valued}\ (R^{-1})]\!] \Longrightarrow ((\in), (\in)) \in R \to \langle R\rangle \mathit{set\text{-}rel} \to \mathit{bool\text{-}rel}$

⟨*proof*⟩

lemma *param-insert*[*param*]:
$(\mathit{insert},\mathit{insert}) \in R \to \langle R\rangle \mathit{set\text{-}rel} \to \langle R\rangle \mathit{set\text{-}rel}$
⟨*proof*⟩

lemma *param-union*[*param*]:
$((\cup), (\cup)) \in \langle R\rangle \mathit{set\text{-}rel} \to \langle R\rangle \mathit{set\text{-}rel} \to \langle R\rangle \mathit{set\text{-}rel}$
⟨*proof*⟩

lemma *param-inter*[*param*]:
assumes *single-valued R*   *single-valued* (R^{-1})
shows $((\cap), (\cap)) \in \langle R\rangle \mathit{set\text{-}rel} \to \langle R\rangle \mathit{set\text{-}rel} \to \langle R\rangle \mathit{set\text{-}rel}$
⟨*proof*⟩

lemma *param-diff*[*param*]:
assumes *single-valued R*   *single-valued* (R^{-1})
shows $((-), (-)) \in \langle R\rangle \mathit{set\text{-}rel} \to \langle R\rangle \mathit{set\text{-}rel} \to \langle R\rangle \mathit{set\text{-}rel}$
⟨*proof*⟩

lemma *param-subseteq*[*param*]:

$\llbracket$*single-valued R*; *single-valued* $(R^{-1})\rrbracket \Longrightarrow ((\subseteq), (\subseteq)) \in \langle R\rangle$*set-rel* $\rightarrow \langle R\rangle$*set-rel* $\rightarrow$ *bool-rel*
⟨*proof*⟩

lemma *param-subset*[*param*]:
$\llbracket$*single-valued R*; *single-valued* $(R^{-1})\rrbracket \Longrightarrow ((\subset), (\subset)) \in \langle R\rangle$*set-rel* $\rightarrow \langle R\rangle$*set-rel* $\rightarrow$ *bool-rel*
⟨*proof*⟩

lemma *param-Ball*[*param*]: (*Ball*,*Ball*)$\in\langle Ra\rangle$*set-rel*$\rightarrow$(*Ra*$\rightarrow$*Id*)$\rightarrow$*Id*
⟨*proof*⟩

lemma *param-Bex*[*param*]: (*Bex*,*Bex*)$\in\langle Ra\rangle$*set-rel*$\rightarrow$(*Ra*$\rightarrow$*Id*)$\rightarrow$*Id*
⟨*proof*⟩

lemma *param-set*[*param*]:
single-valued Ra $\Longrightarrow$ (*set*,*set*)$\in\langle Ra\rangle$*list-rel* $\rightarrow \langle Ra\rangle$*set-rel*
⟨*proof*⟩

lemma *param-Collect*[*param*]:
$\llbracket$*Domain A* = *UNIV*; *Range A* = *UNIV*$\rrbracket \Longrightarrow$ (*Collect*,*Collect*)$\in$(*A*$\rightarrow$*bool-rel*) $\rightarrow$ $\langle A\rangle$*set-rel*
⟨*proof*⟩

lemma *param-finite*[*param*]: $\llbracket$
single-valued R; *single-valued* (R^{-1})
$\rrbracket \Longrightarrow$ (*finite*,*finite*) $\in \langle R\rangle$*set-rel* $\rightarrow$ *bool-rel*
⟨*proof*⟩

lemma *param-card*[*param*]: $\llbracket$*single-valued R*; *single-valued* $(R^{-1})\rrbracket$
$\Longrightarrow$ (*card*, *card*) $\in \langle R\rangle$*set-rel* $\rightarrow$*nat-rel*
⟨*proof*⟩

1.3.2 Standard HOL Constructs

lemma *param-if*[*param*]:
assumes $(c,c')\in Id$
assumes $\llbracket c;c'\rrbracket \Longrightarrow (t,t')\in R$
assumes $\llbracket\neg c;\neg c'\rrbracket \Longrightarrow (e,e')\in R$
shows (*If* c t e, *If* c' t' e')$\in R$
⟨*proof*⟩

lemma *param-Let*[*param*]:
(*Let*,*Let*)$\in$*Ra* $\rightarrow$ (*Ra*$\rightarrow$*Rr*) $\rightarrow$ *Rr*
⟨*proof*⟩

1.3.3 Functions

lemma *param-id*[*param*]: (*id*,*id*)$\in R\rightarrow R$ ⟨*proof*⟩

lemma *param-fun-comp*[*param*]: $((o), (o)) \in (Ra \to Rb) \to (Rc \to Ra) \to Rc \to Rb$
⟨*proof*⟩

lemma *param-fun-upd*[*param*]:
$((=), (=)) \in Ra \to Ra \to Id$
$\Longrightarrow (\mathit{fun\text{-}upd}, \mathit{fun\text{-}upd}) \in (Ra \to Rb) \to Ra \to Rb \to Ra \to Rb$
⟨*proof*⟩

1.3.4 Boolean

lemma *rec-bool-is-case*: $\mathit{old.rec\text{-}bool} = \mathit{case\text{-}bool}$
⟨*proof*⟩

lemma *param-bool*[*param*]:
$(\mathit{True}, \mathit{True}) \in Id$
$(\mathit{False}, \mathit{False}) \in Id$
$(\mathit{conj}, \mathit{conj}) \in Id \to Id \to Id$
$(\mathit{disj}, \mathit{disj}) \in Id \to Id \to Id$
$(\mathit{Not}, \mathit{Not}) \in Id \to Id$
$(\mathit{case\text{-}bool}, \mathit{case\text{-}bool}) \in R \to R \to Id \to R$
$(\mathit{old.rec\text{-}bool}, \mathit{old.rec\text{-}bool}) \in R \to R \to Id \to R$
$((\longleftrightarrow), (\longleftrightarrow)) \in Id \to Id \to Id$
$((\longrightarrow), (\longrightarrow)) \in Id \to Id \to Id$
⟨*proof*⟩

lemma *param-and-cong1*: $[\![\, (a,a') \in \mathit{bool\text{-}rel}; [\![a; a']\!] \Longrightarrow (b,b') \in \mathit{bool\text{-}rel} \,]\!] \Longrightarrow (a \wedge b, a' \wedge b') \in \mathit{bool\text{-}rel}$
⟨*proof*⟩
lemma *param-and-cong2*: $[\![\, (a,a') \in \mathit{bool\text{-}rel}; [\![a; a']\!] \Longrightarrow (b,b') \in \mathit{bool\text{-}rel} \,]\!] \Longrightarrow (b \wedge a, b' \wedge a') \in \mathit{bool\text{-}rel}$
⟨*proof*⟩

1.3.5 Nat

lemma *param-nat1*[*param*]:
$(0,\ 0::\mathit{nat}) \in Id$
$(\mathit{Suc},\ \mathit{Suc}) \in Id \to Id$
$(1,\ 1::\mathit{nat}) \in Id$
$(\mathit{numeral}\ n::\mathit{nat}, \mathit{numeral}\ n::\mathit{nat}) \in Id$
$((<), (<) ::\mathit{nat} \Rightarrow \text{-}) \in Id \to Id \to Id$
$((\leq), (\leq) ::\mathit{nat} \Rightarrow \text{-}) \in Id \to Id \to Id$
$((=), (=) ::\mathit{nat} \Rightarrow \text{-}) \in Id \to Id \to Id$
$((+) ::\mathit{nat} \Rightarrow \text{-}, (+)) \in Id \to Id \to Id$
$((-) ::\mathit{nat} \Rightarrow \text{-}, (-)) \in Id \to Id \to Id$
$((*) ::\mathit{nat} \Rightarrow \text{-}, (*)) \in Id \to Id \to Id$
$((\mathit{div}) ::\mathit{nat} \Rightarrow \text{-}, (\mathit{div})) \in Id \to Id \to Id$
$((\mathit{mod}) ::\mathit{nat} \Rightarrow \text{-}, (\mathit{mod})) \in Id \to Id \to Id$
⟨*proof*⟩

lemma *param-case-nat*[*param*]:
$(\mathit{case\text{-}nat}, \mathit{case\text{-}nat}) \in Ra \to (Id \to Ra) \to Id \to Ra$

$\langle proof \rangle$

lemma *param-rec-nat*[*param*]:
$(\mathit{rec\text{-}nat},\mathit{rec\text{-}nat}) \in R \to (Id \to R \to R) \to Id \to R$
$\langle proof \rangle$

1.3.6 Int

lemma *param-int*[*param*]:
$(0,\ 0::int) \in Id$
$(1,\ 1::int) \in Id$
$(\mathit{numeral}\ n::int,\mathit{numeral}\ n::int) \in Id$
$((<),\ (<)\ ::int \Rightarrow \text{-}) \in Id \to Id \to Id$
$((\leq),\ (\leq)\ ::int \Rightarrow \text{-}) \in Id \to Id \to Id$
$((=),\ (=)\ ::int \Rightarrow \text{-}) \in Id \to Id \to Id$
$((+)\ ::int\Rightarrow\text{-},(+))\in Id\to Id\to Id$
$((-)\ ::int\Rightarrow\text{-},(-))\in Id\to Id\to Id$
$((*)\ ::int\Rightarrow\text{-},(*))\in Id\to Id\to Id$
$((div)\ ::int\Rightarrow\text{-},(div))\in Id\to Id\to Id$
$((mod)\ ::int\Rightarrow\text{-},(mod))\in Id\to Id\to Id$
$\langle proof \rangle$

1.3.7 Product

lemma *param-unit*[*param*]: $((),())\in \mathit{unit\text{-}rel}$ $\langle proof \rangle$

lemma *rec-prod-is-case*: $\mathit{old.rec\text{-}prod} = \mathit{case\text{-}prod}$
$\langle proof \rangle$

lemma *param-prod*[*param*]:
$(Pair,Pair)\in Ra \to Rb \to \langle Ra,Rb\rangle \mathit{prod\text{-}rel}$
$(\mathit{case\text{-}prod},\mathit{case\text{-}prod}) \in (Ra \to Rb \to Rr) \to \langle Ra,Rb\rangle \mathit{prod\text{-}rel} \to Rr$
$(\mathit{old.rec\text{-}prod},\mathit{old.rec\text{-}prod}) \in (Ra \to Rb \to Rr) \to \langle Ra,Rb\rangle \mathit{prod\text{-}rel} \to Rr$
$(fst,fst)\in\langle Ra,Rb\rangle \mathit{prod\text{-}rel} \to Ra$
$(snd,snd)\in\langle Ra,Rb\rangle \mathit{prod\text{-}rel} \to Rb$
$\langle proof \rangle$

lemma *param-case-prod'*:
$[\![\ (p,p')\in\langle Ra,Rb\rangle \mathit{prod\text{-}rel};$
$\bigwedge a\ b\ a'\ b'.\ [\![\ p=(a,b);\ p'=(a',b');\ (a,a')\in Ra;\ (b,b')\in Rb\ ]\!]$
$\Longrightarrow (f\ a\ b,\ f'\ a'\ b')\in R$
$]\!] \Longrightarrow (\mathit{case\text{-}prod}\ f\ p,\ \mathit{case\text{-}prod}\ f'\ p') \in R$
$\langle proof \rangle$

lemma *param-case-prod''*:
$[\![$
$\bigwedge a\ b\ a'\ b'.\ [\![p=(a,b);\ p'=(a',b')]\!] \Longrightarrow (f\ a\ b,f'\ a'\ b')\in R$
$]\!] \Longrightarrow (\mathit{case\text{-}prod}\ f\ p,\ \mathit{case\text{-}prod}\ f'\ p')\in R$
$\langle proof \rangle$

lemma *param-map-prod*[*param*]:
$(\mathit{map\text{-}prod},\ \mathit{map\text{-}prod})$
$\in (Ra \to Rb) \to (Rc \to Rd) \to \langle Ra,Rc\rangle \mathit{prod\text{-}rel} \to \langle Rb,Rd\rangle \mathit{prod\text{-}rel}$
⟨*proof*⟩

lemma *param-apfst*[*param*]:
$(\mathit{apfst},\mathit{apfst}) \in (Ra \to Rb) \to \langle Ra,Rc\rangle \mathit{prod\text{-}rel} \to \langle Rb,Rc\rangle \mathit{prod\text{-}rel}$
⟨*proof*⟩

lemma *param-apsnd*[*param*]:
$(\mathit{apsnd},\mathit{apsnd}) \in (Rb \to Rc) \to \langle Ra,Rb\rangle \mathit{prod\text{-}rel} \to \langle Ra,Rc\rangle \mathit{prod\text{-}rel}$
⟨*proof*⟩

lemma *param-curry*[*param*]:
$(\mathit{curry},\mathit{curry}) \in (\langle Ra,Rb\rangle \mathit{prod\text{-}rel} \to Rc) \to Ra \to Rb \to Rc$
⟨*proof*⟩

lemma *param-uncurry*[*param*]: $(\mathit{uncurry},\mathit{uncurry}) \in (A \to B \to C) \to A \times_r B \to C$
⟨*proof*⟩

lemma *param-prod-swap*[*param*]: $(\mathit{prod.swap},\ \mathit{prod.swap}) \in A \times_r B \to B \times_r A$ ⟨*proof*⟩

context *partial-function-definitions* **begin**
lemma
assumes M: *monotone le-fun le-fun F*
and M': *monotone le-fun le-fun F′*
assumes ADM:
admissible $(\lambda a.\ \forall x\ xa.\ (x,\ xa) \in Rb \longrightarrow (a\ x,\ \mathit{fixp\text{-}fun}\ F'\ xa) \in Ra)$
assumes *bot*: $\bigwedge x\ xa.\ (x,\ xa) \in Rb \Longrightarrow (\mathit{lub}\ \{\},\ \mathit{fixp\text{-}fun}\ F'\ xa) \in Ra$
assumes F: $(F,F') \in (Rb \to Ra) \to Rb \to Ra$
assumes A: $(x,x') \in Rb$
shows $(\mathit{fixp\text{-}fun}\ F\ x,\ \mathit{fixp\text{-}fun}\ F'\ x') \in Ra$
⟨*proof*⟩
end

1.3.8 Option

lemma *param-option*[*param*]:
$(\mathit{None},\mathit{None}) \in \langle R\rangle \mathit{option\text{-}rel}$
$(\mathit{Some},\mathit{Some}) \in R \to \langle R\rangle \mathit{option\text{-}rel}$
$(\mathit{case\text{-}option},\mathit{case\text{-}option}) \in Rr \to (R \to Rr) \to \langle R\rangle \mathit{option\text{-}rel} \to Rr$
$(\mathit{rec\text{-}option},\mathit{rec\text{-}option}) \in Rr \to (R \to Rr) \to \langle R\rangle \mathit{option\text{-}rel} \to Rr$
⟨*proof*⟩

lemma *param-map-option*[*param*]: $(\mathit{map\text{-}option},\ \mathit{map\text{-}option}) \in (A \to B) \to \langle A\rangle \mathit{option\text{-}rel}$
$\to \langle B\rangle \mathit{option\text{-}rel}$
⟨*proof*⟩

lemma *param-case-option'*:
$\llbracket$ $(x,x')\in\langle Rv\rangle$*option-rel*;
$\llbracket x$=*None*; x'=*None* $\rrbracket \Longrightarrow (fn,fn')\in R$;
$\bigwedge v\ v'.$ $\llbracket$ x=*Some* v; x'=*Some* v'; $(v,v')\in Rv$ $\rrbracket \Longrightarrow (fs\ v,\ fs'\ v')\in R$
$\rrbracket \Longrightarrow$ (*case-option* fn fs x, *case-option* fn' fs' x') $\in R$
⟨*proof*⟩

lemma *the-paramL*: $\llbracket l\neq$*None*; $(l,r)\in\langle R\rangle$*option-rel*$\rrbracket \Longrightarrow$ (*the* l, *the* r)$\in R$
⟨*proof*⟩

lemma *the-paramR*: $\llbracket r\neq$*None*; $(l,r)\in\langle R\rangle$*option-rel*$\rrbracket \Longrightarrow$ (*the* l, *the* r)$\in R$
⟨*proof*⟩

lemma *the-default-param*[*param*]:
(*the-default*, *the-default*) $\in R \rightarrow \langle R\rangle$*option-rel* $\rightarrow R$
⟨*proof*⟩

1.3.9 Sum

lemma *rec-sum-is-case*: *old.rec-sum* = *case-sum*
⟨*proof*⟩

lemma *param-sum*[*param*]:
(*Inl*,*Inl*) $\in Rl \rightarrow \langle Rl,Rr\rangle$*sum-rel*
(*Inr*,*Inr*) $\in Rr \rightarrow \langle Rl,Rr\rangle$*sum-rel*
(*case-sum*,*case-sum*) $\in (Rl \rightarrow R) \rightarrow (Rr \rightarrow R) \rightarrow \langle Rl,Rr\rangle$*sum-rel* $\rightarrow R$
(*old.rec-sum*,*old.rec-sum*) $\in (Rl \rightarrow R) \rightarrow (Rr \rightarrow R) \rightarrow \langle Rl,Rr\rangle$*sum-rel* $\rightarrow R$
⟨*proof*⟩

lemma *param-case-sum'*:
$\llbracket$ $(s,s')\in\langle Rl,Rr\rangle$*sum-rel*;
$\bigwedge l\ l'.$ $\llbracket$ s=*Inl* l; s'=*Inl* l'; $(l,l')\in Rl$ $\rrbracket \Longrightarrow (fl\ l,\ fl'\ l')\in R$;
$\bigwedge r\ r'.$ $\llbracket$ s=*Inr* r; s'=*Inr* r'; $(r,r')\in Rr$ $\rrbracket \Longrightarrow (fr\ r,\ fr'\ r')\in R$
$\rrbracket \Longrightarrow$ (*case-sum* fl fr s, *case-sum* fl' fr' s')$\in R$
⟨*proof*⟩

primrec *is-Inl* **where** *is-Inl* (*Inl* -) = *True* | *is-Inl* (*Inr* -) = *False*
primrec *is-Inr* **where** *is-Inr* (*Inr* -) = *True* | *is-Inr* (*Inl* -) = *False*

lemma *is-Inl-param*[*param*]: (*is-Inl*,*is-Inl*) $\in \langle Ra,Rb\rangle$*sum-rel* $\rightarrow$ *bool-rel*
⟨*proof*⟩
lemma *is-Inr-param*[*param*]: (*is-Inr*,*is-Inr*) $\in \langle Ra,Rb\rangle$*sum-rel* $\rightarrow$ *bool-rel*
⟨*proof*⟩

lemma *sum-projl-param*[*param*]:
$\llbracket$*is-Inl* s; $(s',s)\in\langle Ra,Rb\rangle$*sum-rel*$\rrbracket$
$\Longrightarrow$ (*Sum-Type.sum.projl* s',*Sum-Type.sum.projl* s) $\in Ra$
⟨*proof*⟩

lemma *sum-projr-param*[*param*]:
$[\![$*is-Inr s*; $(s',s) \in \langle Ra,Rb \rangle$*sum-rel*$]\!]$
$\Longrightarrow$ (*Sum-Type.sum.projr* s',*Sum-Type.sum.projr* s) $\in$ *Rb*
⟨*proof*⟩

1.3.10 List

lemma *list-rel-append1*: $(as\ @\ bs,\ l) \in \langle R \rangle$*list-rel*
$\longleftrightarrow$ ($\exists$ *cs ds*. $l = cs@ds \wedge (as,cs) \in \langle R \rangle$*list-rel* $\wedge$ $(bs,ds) \in \langle R \rangle$*list-rel*)
⟨*proof*⟩

lemma *list-rel-append2*: $(l,as\ @\ bs) \in \langle R \rangle$*list-rel*
$\longleftrightarrow$ ($\exists$ *cs ds*. $l = cs@ds \wedge (cs,as) \in \langle R \rangle$*list-rel* $\wedge$ $(ds,bs) \in \langle R \rangle$*list-rel*)
⟨*proof*⟩

lemma *param-append*[*param*]:
(*append*, *append*)$\in \langle R \rangle$*list-rel* $\rightarrow$ $\langle R \rangle$*list-rel* $\rightarrow$ $\langle R \rangle$*list-rel*
⟨*proof*⟩

lemma *param-list1*[*param*]:
(*Nil*,*Nil*)$\in \langle R \rangle$*list-rel*
(*Cons*,*Cons*)$\in R \rightarrow \langle R \rangle$*list-rel* $\rightarrow$ $\langle R \rangle$*list-rel*
(*case-list*,*case-list*)$\in Rr \rightarrow (R \rightarrow \langle R \rangle$*list-rel*$\rightarrow Rr) \rightarrow \langle R \rangle$*list-rel*$\rightarrow Rr$
⟨*proof*⟩

lemma *param-rec-list*[*param*]:
(*rec-list*,*rec-list*)
$\in Ra \rightarrow (Rb \rightarrow \langle Rb \rangle$*list-rel* $\rightarrow Ra \rightarrow Ra) \rightarrow \langle Rb \rangle$*list-rel* $\rightarrow Ra$
⟨*proof*⟩

lemma *param-case-list'*:
$[\![$ $(l,l') \in \langle Rb \rangle$*list-rel*;
$[\![l=[];\ l'=[]]\!] \Longrightarrow (n,n') \in Ra$;
$\bigwedge x\ xs\ x'\ xs'.\ [\![\ l=x\#xs;\ l'=x'\#xs';\ (x,x') \in Rb;\ (xs,xs') \in \langle Rb \rangle$*list-rel* $]\!]$
$\Longrightarrow (c\ x\ xs,\ c'\ x'\ xs') \in Ra$
$]\!] \Longrightarrow$ (*case-list* $n\ c\ l$, *case-list* $n'\ c'\ l'$) $\in Ra$
⟨*proof*⟩

lemma *param-map*[*param*]:
$(map,map) \in (R1 \rightarrow R2) \rightarrow \langle R1 \rangle$*list-rel* $\rightarrow \langle R2 \rangle$*list-rel*
⟨*proof*⟩

lemma *param-fold*[*param*]:
$(fold,fold) \in (Re \rightarrow Rs \rightarrow Rs) \rightarrow \langle Re \rangle$*list-rel* $\rightarrow Rs \rightarrow Rs$
$(foldl,foldl) \in (Rs \rightarrow Re \rightarrow Rs) \rightarrow Rs \rightarrow \langle Re \rangle$*list-rel* $\rightarrow Rs$
$(foldr,foldr) \in (Re \rightarrow Rs \rightarrow Rs) \rightarrow \langle Re \rangle$*list-rel* $\rightarrow Rs \rightarrow Rs$
⟨*proof*⟩

lemma *param-list-all*[*param*]: (*list-all*,*list-all*) ∈ (A→*bool-rel*) → ⟨A⟩*list-rel* → *bool-rel*
⟨*proof*⟩

context begin
private primrec *list-all2-alt* :: (′*a* ⇒ ′*b* ⇒ *bool*) ⇒ ′*a list* ⇒ ′*b list* ⇒ *bool* **where**
list-all2-alt P [] *ys* ⟷ (*case ys of* [] ⇒ *True* | - ⇒ *False*)
| *list-all2-alt P* (*x*#*xs*) *ys* ⟷ (*case ys of* [] ⇒ *False* | *y*#*ys* ⇒ *P x y* ∧ *list-all2-alt P xs ys*)

private lemma *list-all2-alt*: *list-all2 P xs ys* = *list-all2-alt P xs ys*
⟨*proof*⟩

lemma *param-list-all2*[*param*]: (*list-all2*, *list-all2*) ∈ (A→B→*bool-rel*) → ⟨A⟩*list-rel* → ⟨B⟩*list-rel* → *bool-rel*
⟨*proof*⟩

end

lemma *param-hd*[*param*]: l≠[] ⟹ (l',l)∈⟨A⟩*list-rel* ⟹ (*hd* l', *hd* l)∈A
⟨*proof*⟩

lemma *param-last*[*param*]:
assumes $y \neq$ []
assumes (x, y) ∈ ⟨A⟩*list-rel*
shows (*last* x, *last* y) ∈ A
⟨*proof*⟩

lemma *param-rotate1*[*param*]: (*rotate1*, *rotate1*) ∈ ⟨A⟩*list-rel* → ⟨A⟩*list-rel*
⟨*proof*⟩

schematic-goal *param-take*[*param*]: (*take*,*take*)∈(*?R*::(-×-) *set*)
⟨*proof*⟩

schematic-goal *param-drop*[*param*]: (*drop*,*drop*)∈(*?R*::(-×-) *set*)
⟨*proof*⟩

schematic-goal *param-length*[*param*]:
(*length*,*length*)∈(*?R*::(-×-) *set*)
⟨*proof*⟩

fun *list-eq* :: (′*a* ⇒ ′*a* ⇒ *bool*) ⇒ ′*a list* ⇒ ′*a list* ⇒ *bool* **where**
list-eq eq [] [] ⟷ *True*
| *list-eq eq* (*a*#*l*) (*a*′#*l*′)
⟷ (*if eq a a*′ *then list-eq eq l l*′ *else False*)
| *list-eq* - - - ⟷ *False*

lemma *param-list-eq*[*param*]:

$(\mathit{list\text{-}eq},\mathit{list\text{-}eq}) \in$
$(R \to R \to \mathit{Id}) \to \langle R\rangle \mathit{list\text{-}rel} \to \langle R\rangle \mathit{list\text{-}rel} \to \mathit{Id}$
⟨*proof*⟩

lemma *id-list-eq-aux*[*simp*]: $(\mathit{list\text{-}eq}\ (=)) = (=)$
⟨*proof*⟩

lemma *param-list-equals*[*param*]:
$[\![\ ((=),\ (=)) \in R {\to} R {\to} \mathit{Id}\ ]\!]$
$\Longrightarrow ((=),\ (=)) \in \langle R\rangle \mathit{list\text{-}rel} \to \langle R\rangle \mathit{list\text{-}rel} \to \mathit{Id}$
⟨*proof*⟩

lemma *param-tl*[*param*]:
$(\mathit{tl},\mathit{tl}) \in \langle R\rangle \mathit{list\text{-}rel} \to \langle R\rangle \mathit{list\text{-}rel}$
⟨*proof*⟩

primrec *list-all-rec* **where**
$\mathit{list\text{-}all\text{-}rec}\ P\ []\longleftrightarrow \mathit{True}$
$|\ \mathit{list\text{-}all\text{-}rec}\ P\ (a\#l) \longleftrightarrow P\ a \wedge \mathit{list\text{-}all\text{-}rec}\ P\ l$

primrec *list-ex-rec* **where**
$\mathit{list\text{-}ex\text{-}rec}\ P\ []\longleftrightarrow \mathit{False}$
$|\ \mathit{list\text{-}ex\text{-}rec}\ P\ (a\#l) \longleftrightarrow P\ a \vee \mathit{list\text{-}ex\text{-}rec}\ P\ l$

lemma *list-all-rec-eq*: $(\forall\, x{\in}\mathit{set}\ l.\ P\ x) = \mathit{list\text{-}all\text{-}rec}\ P\ l$
⟨*proof*⟩

lemma *list-ex-rec-eq*: $(\exists\, x{\in}\mathit{set}\ l.\ P\ x) = \mathit{list\text{-}ex\text{-}rec}\ P\ l$
⟨*proof*⟩

lemma *param-list-ball*[*param*]:
$[\![(P,P'){\in}(\mathit{Ra}{\to}\mathit{Id});\ (l,l'){\in}\langle \mathit{Ra}\rangle\ \mathit{list\text{-}rel}]\!]$
$\Longrightarrow (\forall\, x{\in}\mathit{set}\ l.\ P\ x,\ \forall\, x{\in}\mathit{set}\ l'.\ P'\ x) \in \mathit{Id}$
⟨*proof*⟩

lemma *param-list-bex*[*param*]:
$[\![(P,P'){\in}(\mathit{Ra}{\to}\mathit{Id});\ (l,l'){\in}\langle \mathit{Ra}\rangle\ \mathit{list\text{-}rel}]\!]$
$\Longrightarrow (\exists\, x{\in}\mathit{set}\ l.\ P\ x,\ \exists\, x{\in}\mathit{set}\ l'.\ P'\ x) \in \mathit{Id}$
⟨*proof*⟩

lemma *param-rev*[*param*]: $(\mathit{rev},\mathit{rev}) \in \langle R\rangle \mathit{list\text{-}rel} \to \langle R\rangle \mathit{list\text{-}rel}$
⟨*proof*⟩

lemma *param-foldli*[*param*]: $(\mathit{foldli},\ \mathit{foldli})$
$\in \langle \mathit{Re}\rangle \mathit{list\text{-}rel} \to (\mathit{Rs}{\to}\mathit{Id}) \to (\mathit{Re}{\to}\mathit{Rs}{\to}\mathit{Rs}) \to \mathit{Rs} \to \mathit{Rs}$
⟨*proof*⟩

lemma *param-foldri*[*param*]: $(\mathit{foldri},\ \mathit{foldri})$

$\in \langle Re\rangle list\text{-}rel \rightarrow (Rs{\rightarrow}Id) \rightarrow (Re{\rightarrow}Rs{\rightarrow}Rs) \rightarrow Rs \rightarrow Rs$
⟨*proof*⟩

lemma *param-nth*[*param*]:
assumes *I*: $i'{<}length\ l'$
assumes *IR*: $(i,i'){\in}nat\text{-}rel$
assumes *LR*: $(l,l'){\in}\langle R\rangle list\text{-}rel$
shows $(l!i,l'!i') \in R$
⟨*proof*⟩

lemma *param-replicate*[*param*]:
$(replicate,replicate) \in nat\text{-}rel \rightarrow R \rightarrow \langle R\rangle list\text{-}rel$
⟨*proof*⟩

term *list-update*
lemma *param-list-update*[*param*]:
$(list\text{-}update,list\text{-}update) \in \langle Ra\rangle list\text{-}rel \rightarrow nat\text{-}rel \rightarrow Ra \rightarrow \langle Ra\rangle list\text{-}rel$
⟨*proof*⟩

lemma *param-zip*[*param*]:
$(zip,\ zip) \in \langle Ra\rangle list\text{-}rel \rightarrow \langle Rb\rangle list\text{-}rel \rightarrow \langle\langle Ra,Rb\rangle prod\text{-}rel\rangle list\text{-}rel$
⟨*proof*⟩

lemma *param-upt*[*param*]:
$(upt,\ upt) \in nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow \langle nat\text{-}rel\rangle list\text{-}rel$
⟨*proof*⟩

lemma *param-concat*[*param*]: $(concat,\ concat) \in$
$\langle\langle R\rangle list\text{-}rel\rangle list\text{-}rel \rightarrow \langle R\rangle list\text{-}rel$
⟨*proof*⟩

lemma *param-all-interval-nat*[*param*]:
$(List.all\text{-}interval\text{-}nat,\ List.all\text{-}interval\text{-}nat)$
$\in (nat\text{-}rel \rightarrow bool\text{-}rel) \rightarrow nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow bool\text{-}rel$
⟨*proof*⟩

lemma *param-dropWhile*[*param*]:
$(dropWhile,\ dropWhile) \in (a \rightarrow bool\text{-}rel) \rightarrow \langle a\rangle list\text{-}rel \rightarrow \langle a\rangle list\text{-}rel$
⟨*proof*⟩

lemma *param-takeWhile*[*param*]:
$(takeWhile,\ takeWhile) \in (a \rightarrow bool\text{-}rel) \rightarrow \langle a\rangle list\text{-}rel \rightarrow \langle a\rangle list\text{-}rel$
⟨*proof*⟩

end

Chapter 2

Automatic Refinement

2.1 Automatic Refinement Tool

theory *Autoref-Tool*
imports
  Autoref-Translate
  Autoref-Gen-Algo
  Autoref-Relator-Interface
begin

2.1.1 Standard setup

Declaration of standard phases

⟨*ML*⟩

Main method

⟨*ML*⟩

2.1.2 Tools

⟨*ML*⟩

2.1.3 Advanced Debugging

⟨*ML*⟩

General casting-tag, that allows type-casting on concrete level, while being identity on abstract level.

definition [*simp*]: $CAST \equiv id$
lemma [*autoref-itype*]: $CAST ::_i I \rightarrow_i I$ ⟨*proof*⟩

Hide internal stuff

notation (*input*) *rel-ANNOT* (**infix** ‹$:::_r$› *10*)
notation (*input*) *ind-ANNOT* (**infix** ‹$::\#_r$› *10*)

locale *autoref-syn* **begin**
  notation (*input*) *APP* (**infixl** ‹$› *900*)
  notation (*input*) *rel-ANNOT* (**infix** ‹:::› *10*)
  notation (*input*) *ind-ANNOT* (**infix** ‹::#› *10*)
  notation *OP* (‹*OP*›)
  notation (*input*) *ABS* (**binder** ‹λ′′› *10*)
end

no-notation (*input*) *APP* (**infixl** ‹$› *900*)
no-notation (*input*) *ABS* (**binder** ‹λ′′› *10*)

no-notation (*input*) *rel-ANNOT* (**infix** ‹:::› *10*)
no-notation (*input*) *ind-ANNOT* (**infix** ‹::#› *10*)

hide-const (**open**) *PROTECT ANNOT OP APP ABS ID-FAIL rel-annot ind-annot*

end

2.2 Standard HOL Bindings

theory *Autoref-Bindings-HOL*
imports *Tool/Autoref-Tool*
begin

2.2.1 Structural Expansion

In some situations, autoref imitates the operations on typeclasses and the typeclass hierarchy. This may result in structural mismatches, e.g., a hash-code side-condition may look like:

is-hashcode (*prod-eq* (=) (=)) *hashcode*

This cannot be discharged by the rule

is-hashcode (=) *hashcode*

In order to handle such cases, we introduce a set of simplification lemmas that expand the structure of an operator as far as possible. These lemmas are integrated into a tagged solver, that can prove equality between operators modulo structural expansion.

definition [*simp*]: *STRUCT-EQ-tag x y* ≡ *x* = *y*
lemma *STRUCT-EQ-tagI*: *x*=*y* ⟹ *STRUCT-EQ-tag x y* ⟨*proof*⟩

⟨*ML*⟩

Sometimes, also relators must be expanded. Usually to check them to be the identity relator

definition [*simp*]: *REL-IS-ID R* ≡ *R=Id*
definition [*simp*]: *REL-FORCE-ID R* ≡ *R=Id*
lemma *REL-IS-ID-trigger*: *R=Id* ⟹ *REL-IS-ID R* ⟨*proof*⟩
lemma *REL-FORCE-ID-trigger*: *R=Id* ⟹ *REL-FORCE-ID R* ⟨*proof*⟩

⟨*ML*⟩

abbreviation *PREFER-id R* ≡ *PREFER REL-IS-ID R*

lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of fun-rel i-fun*]

2.2.2 Booleans

consts
i-bool :: *interface*

lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of bool-rel i-bool*]

lemma [*autoref-itype*]:
$True ::_i i\text{-}bool$
$False ::_i i\text{-}bool$
$conj ::_i i\text{-}bool \to_i i\text{-}bool \to_i i\text{-}bool$
$(\longleftrightarrow) ::_i i\text{-}bool \to_i i\text{-}bool \to_i i\text{-}bool$
$(\longrightarrow) ::_i i\text{-}bool \to_i i\text{-}bool \to_i i\text{-}bool$
$disj ::_i i\text{-}bool \to_i i\text{-}bool \to_i i\text{-}bool$
$Not ::_i i\text{-}bool \to_i i\text{-}bool$
$case\text{-}bool ::_i I \to_i I \to_i i\text{-}bool \to_i I$
$old.rec\text{-}bool ::_i I \to_i I \to_i i\text{-}bool \to_i I$
⟨*proof*⟩

lemma *autoref-bool*[*autoref-rules*]:
$(x,x) \in bool\text{-}rel$
$(conj,conj) \in bool\text{-}rel \to bool\text{-}rel \to bool\text{-}rel$
$(disj,disj) \in bool\text{-}rel \to bool\text{-}rel \to bool\text{-}rel$
$(Not,Not) \in bool\text{-}rel \to bool\text{-}rel$
$(case\text{-}bool,case\text{-}bool) \in R \to R \to bool\text{-}rel \to R$
$(old.rec\text{-}bool,old.rec\text{-}bool) \in R \to R \to bool\text{-}rel \to R$
$((\longleftrightarrow), (\longleftrightarrow)) \in bool\text{-}rel \to bool\text{-}rel \to bool\text{-}rel$
$((\longrightarrow), (\longrightarrow)) \in bool\text{-}rel \to bool\text{-}rel \to bool\text{-}rel$
⟨*proof*⟩

2.2.3 Standard Type Classes

context begin interpretation *autoref-syn* ⟨*proof*⟩

We allow these operators for all interfaces.

lemma [*autoref-itype*]:
$(<) ::_i I \rightarrow_i I \rightarrow_i$ *i-bool*
$(\leq) ::_i I \rightarrow_i I \rightarrow_i$ *i-bool*
$(=) ::_i I \rightarrow_i I \rightarrow_i$ *i-bool*
$(+) ::_i I \rightarrow_i I \rightarrow_i I$
$(-) ::_i I \rightarrow_i I \rightarrow_i I$
$(div) ::_i I \rightarrow_i I \rightarrow_i I$
$(mod) ::_i I \rightarrow_i I \rightarrow_i I$
$(*) ::_i I \rightarrow_i I \rightarrow_i I$
$0 ::_i I$
$1 ::_i I$
$numeral\ x ::_i I$
$uminus ::_i I \rightarrow_i I$
$\langle proof \rangle$

lemma *pat-num-generic*[*autoref-op-pat*]:
$0 \equiv OP\ 0 :::_i I$
$1 \equiv OP\ 1 :::_i I$
$numeral\ x \equiv (OP\ (numeral\ x) :::_i I)$
$\langle proof \rangle$

lemma [*autoref-rules*]:
assumes *PRIO-TAG-GEN-ALGO*
shows $((<), (<)) \in Id \rightarrow Id \rightarrow bool\text{-}rel$
and $((\leq), (\leq)) \in Id \rightarrow Id \rightarrow bool\text{-}rel$
and $((=), (=)) \in Id \rightarrow Id \rightarrow bool\text{-}rel$
and $(numeral\ x, OP\ (numeral\ x) ::: Id) \in Id$
and $(uminus, uminus) \in Id \rightarrow Id$
and $(0,0) \in Id$
and $(1,1) \in Id$
$\langle proof \rangle$

2.2.4 Functional Combinators

lemma [*autoref-itype*]: $id ::_i I \rightarrow_i I$ $\langle proof \rangle$
lemma *autoref-id*[*autoref-rules*]: $(id,id) \in R \rightarrow R$ $\langle proof \rangle$

term (o)
lemma [*autoref-itype*]: $(\circ) ::_i (Ia \rightarrow_i Ib) \rightarrow_i (Ic \rightarrow_i Ia) \rightarrow_i Ic \rightarrow_i Ib$
$\langle proof \rangle$
lemma *autoref-comp*[*autoref-rules*]:
$((o), (o)) \in (Ra \rightarrow Rb) \rightarrow (Rc \rightarrow Ra) \rightarrow Rc \rightarrow Rb$
$\langle proof \rangle$

lemma [*autoref-itype*]: $If ::_i i\text{-}bool \rightarrow_i I \rightarrow_i I \rightarrow_i I$ $\langle proof \rangle$
lemma *autoref-If*[*autoref-rules*]: $(If,If) \in Id \rightarrow R \rightarrow R \rightarrow R$ $\langle proof \rangle$
lemma *autoref-If-cong*[*autoref-rules*]:
assumes $(c',c) \in Id$

assumes *REMOVE-INTERNAL c* $\Longrightarrow$ $(t',t) \in R$
assumes $\neg$ *REMOVE-INTERNAL c* $\Longrightarrow$ $(e',e) \in R$
shows $(If\ c'\ t'\ e', (OP\ If\ :::\ Id \rightarrow R \rightarrow R \rightarrow R)\$c\$t\$e) \in R$
⟨*proof*⟩

lemma [*autoref-itype*]: $Let ::_i Ix \rightarrow_i (Ix \rightarrow_i Iy) \rightarrow_i Iy$ ⟨*proof*⟩
lemma *autoref-Let*:
$(Let, Let) \in Ra \rightarrow (Ra \rightarrow Rr) \rightarrow Rr$
⟨*proof*⟩

lemma *autoref-Let-cong*[*autoref-rules*]:
assumes $(x',x) \in Ra$
assumes $\bigwedge y\ y'.$ *REMOVE-INTERNAL* $(x=y) \Longrightarrow (y',y) \in Ra \Longrightarrow (f'\ y', f\$y) \in Rr$
shows $(Let\ x'\ f', (OP\ Let\ :::\ Ra \rightarrow (Ra \rightarrow Rr) \rightarrow Rr)\$x\$f) \in Rr$
⟨*proof*⟩

end

2.2.5 Unit

consts *i-unit* :: *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of unit-rel i-unit*]

lemma [*autoref-rules*]: $((),()) \in unit\text{-}rel$ ⟨*proof*⟩

2.2.6 Nat

consts *i-nat* :: *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of nat-rel i-nat*]

context begin interpretation *autoref-syn* ⟨*proof*⟩
lemma *pat-num-nat*[*autoref-op-pat*]:
$0::nat \equiv OP\ 0\ :::_i\ i\text{-}nat$
$1::nat \equiv OP\ 1\ :::_i\ i\text{-}nat$
$(numeral\ x)::nat \equiv (OP\ (numeral\ x)\ :::_i\ i\text{-}nat)$
⟨*proof*⟩

lemma *autoref-nat*[*autoref-rules*]:
$(0,\ 0::nat) \in nat\text{-}rel$
$(Suc,\ Suc) \in nat\text{-}rel \rightarrow nat\text{-}rel$
$(1,\ 1::nat) \in nat\text{-}rel$
$(numeral\ n::nat, numeral\ n::nat) \in nat\text{-}rel$
$((<),\ (<)\ ::nat \Rightarrow \text{-}) \in nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow bool\text{-}rel$
$((\le),\ (\le)\ ::nat \Rightarrow \text{-}) \in nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow bool\text{-}rel$
$((=),\ (=)\ ::nat \Rightarrow \text{-}) \in nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow bool\text{-}rel$
$((+)\ ::nat \Rightarrow \text{-}, (+)) \in nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow nat\text{-}rel$
$((-)\ ::nat \Rightarrow \text{-}, (-)) \in nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow nat\text{-}rel$
$((div)\ ::nat \Rightarrow \text{-}, (div)) \in nat\text{-}rel \rightarrow nat\text{-}rel \rightarrow nat\text{-}rel$

$((*),\ (*)) \in$ *nat-rel*$\rightarrow$*nat-rel*$\rightarrow$*nat-rel*
$((mod),\ (mod)) \in$ *nat-rel*$\rightarrow$*nat-rel*$\rightarrow$*nat-rel*
⟨*proof*⟩

lemma *autoref-case-nat*[*autoref-rules*]:
(*case-nat*,*case-nat*)$\in$*Ra* $\rightarrow$ (*Id* $\rightarrow$ *Ra*) $\rightarrow$ *Id* $\rightarrow$ *Ra*
⟨*proof*⟩

lemma *autoref-rec-nat*: (*rec-nat*,*rec-nat*) $\in$ *R* $\rightarrow$ (*Id* $\rightarrow$ *R* $\rightarrow$ *R*) $\rightarrow$ *Id* $\rightarrow$ *R*
⟨*proof*⟩
end

2.2.7 Int

consts *i-int* :: *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of int-rel i-int*]

context begin interpretation *autoref-syn* ⟨*proof*⟩
lemma *pat-num-int*[*autoref-op-pat*]:
0::*int* $\equiv$ *OP 0* $:::_i$ *i-int*
1::*int* $\equiv$ *OP 1* $:::_i$ *i-int*
(*numeral x*)::*int* $\equiv$ (*OP* (*numeral x*) $:::_i$ *i-int*)
⟨*proof*⟩

lemma *autoref-int*[*autoref-rules* (**overloaded**)]:
(*0*, *0*::*int*) $\in$ *int-rel*
(*1*, *1*::*int*) $\in$ *int-rel*
(*numeral n*::*int*,*numeral n*::*int*) $\in$ *int-rel*
$((<),\ (<)$ ::*int* $\Rightarrow$ -) $\in$ *int-rel* $\rightarrow$ *int-rel* $\rightarrow$ *bool-rel*
$((\leq),\ (\leq)$ ::*int* $\Rightarrow$ -) $\in$ *int-rel* $\rightarrow$ *int-rel* $\rightarrow$ *bool-rel*
$((=),\ (=)$ ::*int* $\Rightarrow$ -) $\in$ *int-rel* $\rightarrow$ *int-rel* $\rightarrow$ *bool-rel*
$((+)$ ::*int*$\Rightarrow$-,$(+))\in$*int-rel*$\rightarrow$*int-rel*$\rightarrow$*int-rel*
$((-)$ ::*int*$\Rightarrow$-,$(-))\in$*int-rel*$\rightarrow$*int-rel*$\rightarrow$*int-rel*
((*div*) ::*int*$\Rightarrow$-,(*div*))$\in$*int-rel*$\rightarrow$*int-rel*$\rightarrow$*int-rel*
(*uminus*,*uminus*)$\in$*int-rel*$\rightarrow$*int-rel*
$((*),\ (*)) \in$ *int-rel*$\rightarrow$*int-rel*$\rightarrow$*int-rel*
$((mod),\ (mod)) \in$ *int-rel*$\rightarrow$*int-rel*$\rightarrow$*int-rel*
⟨*proof*⟩
end

2.2.8 Product

consts *i-prod* :: *interface* $\Rightarrow$ *interface* $\Rightarrow$ *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of prod-rel i-prod*]

context begin interpretation *autoref-syn* ⟨*proof*⟩

lemma *prod-refine*[*autoref-rules*]:
$(Pair,Pair){\in}Ra \rightarrow Rb \rightarrow \langle Ra,Rb\rangle prod\text{-}rel$
$(case\text{-}prod,case\text{-}prod) \in (Ra \rightarrow Rb \rightarrow Rr) \rightarrow \langle Ra,Rb\rangle prod\text{-}rel \rightarrow Rr$
$(old.rec\text{-}prod,old.rec\text{-}prod) \in (Ra \rightarrow Rb \rightarrow Rr) \rightarrow \langle Ra,Rb\rangle prod\text{-}rel \rightarrow Rr$
$(fst,fst){\in}\langle Ra,Rb\rangle prod\text{-}rel \rightarrow Ra$
$(snd,snd){\in}\langle Ra,Rb\rangle prod\text{-}rel \rightarrow Rb$
⟨*proof*⟩

definition *prod-eq eqa eqb x1 x2* ≡
case x1 of (a1,b1) ⇒ case x2 of (a2,b2) ⇒ eqa a1 a2 ∧ eqb b1 b2

lemma *prod-eq-autoref*[*autoref-rules* (**overloaded**)]:
$[\![GEN\text{-}OP\ eqa\ (=)\ (Ra{\rightarrow}Ra{\rightarrow}Id);\ GEN\text{-}OP\ eqb\ (=)\ (Rb{\rightarrow}Rb{\rightarrow}Id)]\!]$
$\Longrightarrow (prod\text{-}eq\ eqa\ eqb,(=)) \in \langle Ra,Rb\rangle prod\text{-}rel \rightarrow \langle Ra,Rb\rangle prod\text{-}rel \rightarrow Id$
⟨*proof*⟩

lemma *prod-eq-expand*[*autoref-struct-expand*]: (=) = *prod-eq* (=) (=)
⟨*proof*⟩
end

2.2.9 Option

consts *i-option* :: *interface* ⇒ *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of option-rel i-option*]

context begin interpretation *autoref-syn* ⟨*proof*⟩

lemma *autoref-opt*[*autoref-rules*]:
$(None,None){\in}\langle R\rangle option\text{-}rel$
$(Some,Some){\in}R \rightarrow \langle R\rangle option\text{-}rel$
$(case\text{-}option,case\text{-}option){\in}Rr{\rightarrow}(R \rightarrow Rr){\rightarrow}\langle R\rangle option\text{-}rel \rightarrow Rr$
$(rec\text{-}option,rec\text{-}option){\in}Rr{\rightarrow}(R \rightarrow Rr){\rightarrow}\langle R\rangle option\text{-}rel \rightarrow Rr$
⟨*proof*⟩

lemma *autoref-the*[*autoref-rules*]:
assumes *SIDE-PRECOND* ($x{\neq}None$)
assumes $(x',x){\in}\langle R\rangle option\text{-}rel$
shows $(the\ x',\ (OP\ the\ :::\ \langle R\rangle option\text{-}rel \rightarrow R)\$x) \in R$
⟨*proof*⟩

lemma *autoref-the-default*[*autoref-rules*]:
$(the\text{-}default,\ the\text{-}default) \in R \rightarrow \langle R\rangle option\text{-}rel \rightarrow R$
⟨*proof*⟩

definition [*simp*]: *is-None a* ≡ *case a of None* ⇒ *True* | - ⇒ *False*
lemma *pat-isNone*[*autoref-op-pat*]:
$a{=}None \equiv (OP\ is\text{-}None\ :::_i\ \langle I\rangle_i i\text{-}option \rightarrow_i\ i\text{-}bool)\$a$
$None{=}a \equiv (OP\ is\text{-}None\ :::_i\ \langle I\rangle_i i\text{-}option \rightarrow_i\ i\text{-}bool)\$a$

⟨*proof*⟩
lemma *autoref-is-None*[*param*,*autoref-rules*]:
(*is-None*,*is-None*)∈⟨*R*⟩*option-rel* → *Id*
⟨*proof*⟩

lemma *fold-is-None*: *x*=*None* ⟷ *is-None x* ⟨*proof*⟩

definition *option-eq eq v1 v2* ≡
case (*v1*,*v2*) *of*
(*None*,*None*) ⇒ *True*
| (*Some x1*, *Some x2*) ⇒ *eq x1 x2*
| - ⇒ *False*

lemma *option-eq-autoref*[*autoref-rules* (**overloaded**)]:
⟦*GEN-OP eq* (=) (*R*→*R*→*Id*)⟧
⟹ (*option-eq eq*,(=)) ∈ ⟨*R*⟩*option-rel* → ⟨*R*⟩*option-rel* → *Id*
⟨*proof*⟩

lemma *option-eq-expand*[*autoref-struct-expand*]:
(=) = *option-eq* (=)
⟨*proof*⟩
end

2.2.10 Sum-Types

consts *i-sum* :: *interface* ⇒ *interface* ⇒ *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of sum-rel i-sum*]

context begin interpretation *autoref-syn* ⟨*proof*⟩

lemma *autoref-sum*[*autoref-rules*]:
(*Inl*,*Inl*) ∈ *Rl* → ⟨*Rl*,*Rr*⟩*sum-rel*
(*Inr*,*Inr*) ∈ *Rr* → ⟨*Rl*,*Rr*⟩*sum-rel*
(*case-sum*,*case-sum*) ∈ (*Rl* → *R*) → (*Rr* → *R*) → ⟨*Rl*,*Rr*⟩*sum-rel* → *R*
(*old.rec-sum*,*old.rec-sum*) ∈ (*Rl* → *R*) → (*Rr* → *R*) → ⟨*Rl*,*Rr*⟩*sum-rel* → *R*
⟨*proof*⟩

definition *sum-eq eql eqr s1 s2* ≡
case (*s1*,*s2*) *of*
(*Inl x1*, *Inl x2*) ⇒ *eql x1 x2*
| (*Inr x1*, *Inr x2*) ⇒ *eqr x1 x2*
| - ⇒ *False*

lemma *sum-eq-autoref*[*autoref-rules* (**overloaded**)]:
⟦*GEN-OP eql* (=) (*Rl*→*Rl*→*Id*); *GEN-OP eqr* (=) (*Rr*→*Rr*→*Id*)⟧
⟹ (*sum-eq eql eqr*,(=)) ∈ ⟨*Rl*,*Rr*⟩*sum-rel* → ⟨*Rl*,*Rr*⟩*sum-rel* → *Id*
⟨*proof*⟩

lemma *sum-eq-expand*[*autoref-struct-expand*]: (=) = *sum-eq* (=) (=)
⟨*proof*⟩

lemmas [*autoref-rules*] = *is-Inl-param is-Inr-param*

lemma *autoref-sum-Projl*[*autoref-rules*]:
⟦*SIDE-PRECOND* (*is-Inl s*); (*s*′,*s*)∈⟨*Ra*,*Rb*⟩*sum-rel*⟧
⟹ (*Sum-Type.sum.projl s*′, (*OP Sum-Type.sum.projl* ::: ⟨*Ra*,*Rb*⟩*sum-rel* → *Ra*)$*s*)∈*Ra*
⟨*proof*⟩

lemma *autoref-sum-Projr*[*autoref-rules*]:
⟦*SIDE-PRECOND* (*is-Inr s*); (*s*′,*s*)∈⟨*Ra*,*Rb*⟩*sum-rel*⟧
⟹ (*Sum-Type.sum.projr s*′, (*OP Sum-Type.sum.projr* ::: ⟨*Ra*,*Rb*⟩*sum-rel* → *Rb*)$*s*)∈*Rb*
⟨*proof*⟩

end

2.2.11 List

consts *i-list* :: *interface* ⇒ *interface*
lemmas [*autoref-rel-intf*] = *REL-INTFI*[*of list-rel i-list*]

context begin interpretation *autoref-syn* ⟨*proof*⟩

lemma *autoref-append*[*autoref-rules*]:
(*append*, *append*)∈⟨*R*⟩*list-rel* → ⟨*R*⟩*list-rel* → ⟨*R*⟩*list-rel*
⟨*proof*⟩

lemma *refine-list*[*autoref-rules*]:
(*Nil*,*Nil*)∈⟨*R*⟩*list-rel*
(*Cons*,*Cons*)∈*R* → ⟨*R*⟩*list-rel* → ⟨*R*⟩*list-rel*
(*case-list*,*case-list*)∈*Rr*→(*R*→⟨*R*⟩*list-rel*→*Rr*)→⟨*R*⟩*list-rel*→*Rr*
⟨*proof*⟩

lemma *autoref-rec-list*[*autoref-rules*]: (*rec-list*,*rec-list*)
∈ *Ra* → (*Rb* → ⟨*Rb*⟩*list-rel* → *Ra* → *Ra*) → ⟨*Rb*⟩*list-rel* → *Ra*
⟨*proof*⟩

lemma *refine-map*[*autoref-rules*]:
(*map*,*map*)∈(*R1*→*R2*) → ⟨*R1*⟩*list-rel* → ⟨*R2*⟩*list-rel*
⟨*proof*⟩

lemma *refine-fold*[*autoref-rules*]:
(*fold*,*fold*)∈(*Re*→*Rs*→*Rs*) → ⟨*Re*⟩*list-rel* → *Rs* → *Rs*
(*foldl*,*foldl*)∈(*Rs*→*Re*→*Rs*) → *Rs* → ⟨*Re*⟩*list-rel* → *Rs*
(*foldr*,*foldr*)∈(*Re*→*Rs*→*Rs*) → ⟨*Re*⟩*list-rel* → *Rs* → *Rs*

⟨*proof*⟩

schematic-goal *autoref-take*[*autoref-rules*]: (*take*,*take*)∈(*?R*::(-×-) *set*)
⟨*proof*⟩
schematic-goal *autoref-drop*[*autoref-rules*]: (*drop*,*drop*)∈(*?R*::(-×-) *set*)
⟨*proof*⟩
schematic-goal *autoref-length*[*autoref-rules*]:
(*length*,*length*)∈(*?R*::(-×-) *set*)
⟨*proof*⟩

lemma *autoref-nth*[*autoref-rules*]:
assumes $(l,l')\in\langle R\rangle \mathit{list\text{-}rel}$
assumes $(i,i')\in \mathit{Id}$
assumes *SIDE-PRECOND* $(i' < \mathit{length}\ l')$
shows $(\mathit{nth}\ l\ i,(\mathit{OP}\ \mathit{nth}\ :::\ \langle R\rangle \mathit{list\text{-}rel} \rightarrow \mathit{Id} \rightarrow R)\$l'\$i')\in R$
⟨*proof*⟩

fun *list-eq* :: $('a \Rightarrow 'a \Rightarrow \mathit{bool}) \Rightarrow 'a\ \mathit{list} \Rightarrow 'a\ \mathit{list} \Rightarrow \mathit{bool}$ **where**
list-eq eq [] [] ⟷ *True*
| *list-eq eq* $(a\#l)\ (a'\#l')$
⟷ (*if eq a* a' *then list-eq eq l* l' *else False*)
| *list-eq* - - - ⟷ *False*

lemma *autoref-list-eq-aux*:
(*list-eq*,*list-eq*) ∈
$(R \rightarrow R \rightarrow \mathit{Id}) \rightarrow \langle R\rangle \mathit{list\text{-}rel} \rightarrow \langle R\rangle \mathit{list\text{-}rel} \rightarrow \mathit{Id}$
⟨*proof*⟩

lemma *list-eq-expand*[*autoref-struct-expand*]: (=) = (*list-eq* (=))
⟨*proof*⟩

lemma *autoref-list-eq*[*autoref-rules* (**overloaded**)]:
GEN-OP eq (=) $(R\rightarrow R\rightarrow \mathit{Id}) \Longrightarrow$ (*list-eq eq*, (=))
$\in \langle R\rangle \mathit{list\text{-}rel} \rightarrow \langle R\rangle \mathit{list\text{-}rel} \rightarrow \mathit{Id}$
⟨*proof*⟩

lemma *autoref-hd*[*autoref-rules*]:
$[\![$ *SIDE-PRECOND* $(l'\neq[\,])$; $(l,l') \in \langle R\rangle \mathit{list\text{-}rel}$ $]\!] \Longrightarrow$
$(\mathit{hd}\ l,(\mathit{OP}\ \mathit{hd}\ :::\ \langle R\rangle \mathit{list\text{-}rel} \rightarrow R)\$l') \in R$
⟨*proof*⟩

lemma *autoref-tl*[*autoref-rules*]:
$(\mathit{tl},\mathit{tl}) \in \langle R\rangle \mathit{list\text{-}rel} \rightarrow \langle R\rangle \mathit{list\text{-}rel}$
⟨*proof*⟩

definition [*simp*]: *is-Nil a* ≡ *case a of* [] ⇒ *True* | - ⇒ *False*

lemma *is-Nil-pat*[*autoref-op-pat*]:
$a=[\,] \equiv (\mathit{OP}\ \mathit{is\text{-}Nil}\ :::_i\ \langle I\rangle_i \mathit{i\text{-}list} \rightarrow_i \mathit{i\text{-}bool})\$a$

$[]{=}a \equiv (OP\ is\text{-}Nil\ :::_i\ \langle I\rangle_i i\text{-}list \rightarrow_i i\text{-}bool)\$a$
⟨proof⟩

lemma *autoref-is-Nil*[*param*,*autoref-rules*]:
$(is\text{-}Nil,is\text{-}Nil) \in \langle R\rangle list\text{-}rel \rightarrow bool\text{-}rel$
⟨proof⟩

lemma *conv-to-is-Nil*:
$l{=}[] \longleftrightarrow is\text{-}Nil\ l$
$[]{=}l \longleftrightarrow is\text{-}Nil\ l$
⟨proof⟩

lemma *autoref-butlast*[*param*, *autoref-rules*]:
$(butlast,butlast) \in \langle R\rangle list\text{-}rel \rightarrow \langle R\rangle list\text{-}rel$
⟨proof⟩

definition [*simp*]: $op\text{-}list\text{-}singleton\ x \equiv [x]$
lemma *op-list-singleton-pat*[*autoref-op-pat*]:
$[x] \equiv (OP\ op\text{-}list\text{-}singleton\ :::_i\ I \rightarrow_i \langle I\rangle_i i\text{-}list)\$x$ ⟨proof⟩
lemma *autoref-list-singleton*[*autoref-rules*]:
$(\lambda a.\ [a],op\text{-}list\text{-}singleton) \in R \rightarrow \langle R\rangle list\text{-}rel$
⟨proof⟩

definition [*simp*]: $op\text{-}list\text{-}append\text{-}elem\ s\ x \equiv s@[x]$

lemma *pat-list-append-elem*[*autoref-op-pat*]:
$s@[x] \equiv (OP\ op\text{-}list\text{-}append\text{-}elem\ :::_i\ \langle I\rangle_i i\text{-}list \rightarrow_i I \rightarrow_i \langle I\rangle_i i\text{-}list)\$s\$x$
⟨proof⟩

lemma *autoref-list-append-elem*[*autoref-rules*]:
$(\lambda s\ x.\ s@[x],\ op\text{-}list\text{-}append\text{-}elem) \in \langle R\rangle list\text{-}rel \rightarrow R \rightarrow \langle R\rangle list\text{-}rel$
⟨proof⟩

declare *param-rev*[*autoref-rules*]

declare *param-all-interval-nat*[*autoref-rules*]
lemma [*autoref-op-pat*]:
$(\forall i<u.\ P\ i) \equiv OP\ List.all\text{-}interval\text{-}nat\ P\ 0\ u$
$(\forall i\le u.\ P\ i) \equiv OP\ List.all\text{-}interval\text{-}nat\ P\ 0\ (Suc\ u)$
$(\forall i<u.\ l\le i \longrightarrow P\ i) \equiv OP\ List.all\text{-}interval\text{-}nat\ P\ l\ u$
$(\forall i\le u.\ l\le i \longrightarrow P\ i) \equiv OP\ List.all\text{-}interval\text{-}nat\ P\ l\ (Suc\ u)$
⟨proof⟩

lemmas [*autoref-rules*] = *param-dropWhile param-takeWhile*

end

2.2.12 Examples

Be careful to make the concrete type a schematic type variable. The default behaviour of *schematic-lemma* makes it a fixed variable, that will not unify with the infered term!

schematic-goal
$(?f::?'c,[1,2,3]@[4::nat])\in ?R$
$\langle proof\rangle$

schematic-goal
$(?f::?'c,[1::nat,$
$2,3,4,5,6,7,8,9,0,1,43,5,5,435,5,1,5,6,5,6,5,63,56$
$]$
$)\in ?R$
$\langle proof\rangle$

schematic-goal
$(?f::?'c,[1,2,3] = [])\in ?R$
$\langle proof\rangle$

When specifying custom refinement rules on the fly, be careful with the type-inference between *notes* and *shows*. It's too easy to „decouple" the type $'a$ in the autoref-rule and the actual goal, as shown below!

schematic-goal
notes $[autoref\text{-}rules] = IdI[$**where** $'a='a]$
notes $[autoref\text{-}itype] = itypeI[$**where** $'t='a::numeral$ **and** $I=i\text{-}std]$
shows $(?f::?'c,\ hd\ [a,b,c::'a::numeral])\in ?R$

The autoref-rule is bound with type $'a::typ$, while the goal statement has $'a::numeral$!

$\langle proof\rangle$

Here comes the correct version. Note the duplicate sort annotation of type $'a$:

schematic-goal
notes $[autoref\text{-}rules\text{-}raw] = IdI[$**where** $'a='a::numeral]$
notes $[autoref\text{-}itype] = itypeI[$**where** $'t='a::numeral$ **and** $I=i\text{-}std]$
shows $(?f::?'c,\ hd\ [a,b,c::'a::numeral])\in ?R$
$\langle proof\rangle$

Special cases of equality: Note that we do not require equality on the element type!

schematic-goal

assumes $[autoref\text{-}rules]$: $(ai,a)\in\langle R\rangle option\text{-}rel$
shows $(?f::?'c,\ a = None)\in ?R$
$\langle proof\rangle$

schematic-goal

  assumes [*autoref-rules*]: $(ai,a){\in}\langle R\rangle list\text{-}rel$
  shows $(?f{::}?'c,\ [] = a){\in}?R$
  ⟨*proof*⟩

schematic-goal
  shows $(?f{::}?'c,\ [1,2] = [2,3{::}nat]){\in}?R$
  ⟨*proof*⟩

end

2.3 Entry Point for the Automatic Refinement Tool

theory *Automatic-Refinement*
imports
  Tool/Autoref-Tool
  Autoref-Bindings-HOL
begin

The automatic refinement tool should be used by importing this theory

2.3.1 Convenience

The following lemmas can be used to add tags to theorems

lemma *PREFER-I*: $P\ x \Longrightarrow PREFER\ P\ x$ ⟨*proof*⟩
lemma *PREFER-D*: $PREFER\ P\ x \Longrightarrow P\ x$ ⟨*proof*⟩

lemmas *PREFER-sv-D* = *PREFER-D*[*of single-valued*]
lemma *PREFER-id-D*: $PREFER\text{-}id\ R \Longrightarrow R{=}Id$ ⟨*proof*⟩

abbreviation $PREFER\text{-}RUNIV \equiv PREFER\ (\lambda R.\ Range\ R = UNIV)$
lemmas *PREFER-RUNIV-D* = *PREFER-D*[*of* $(\lambda R.\ Range\ R = UNIV)$]

lemma *SIDE-GEN-ALGO-D*: $SIDE\text{-}GEN\text{-}ALGO\ P \Longrightarrow P$ ⟨*proof*⟩

lemma *GEN-OP-D*: $GEN\text{-}OP\ c\ a\ R \Longrightarrow (c,a){\in}R$
  ⟨*proof*⟩

lemma *MINOR-PRIO-TAG-I*: $P \Longrightarrow (MINOR\text{-}PRIO\text{-}TAG\ p \Longrightarrow P)$ ⟨*proof*⟩
lemma *MAJOR-PRIO-TAG-I*: $P \Longrightarrow (MAJOR\text{-}PRIO\text{-}TAG\ p \Longrightarrow P)$ ⟨*proof*⟩
lemma *PRIO-TAG-I*: $P \Longrightarrow (PRIO\text{-}TAG\ ma\ mi \Longrightarrow P)$ ⟨*proof*⟩

end