

Transfer

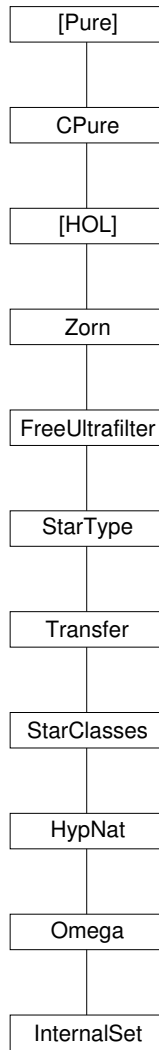
Brian Huffman

May 9, 2005

Contents

1	Zorn's Lemma	4
1.1	Mathematical Preamble	4
1.2	Hausdorff's Theorem: Every Set Contains a Maximal Chain.	5
1.3	Zorn's Lemma: If All Chains Have Upper Bounds Then There Is a Maximal Element	6
1.4	Alternative version of Zorn's Lemma	6
2	Filters and Ultrafilters	7
2.1	Definitions and basic properties	7
2.1.1	Filters	7
2.1.2	Ultrafilters	7
2.1.3	Free ultrafilters	7
2.2	Maximal filter = ultrafilter	8
2.3	Ultrafilter Theorem	8
2.3.1	Unions of chains of superfrechets	9
2.3.2	Existence of free ultrafilter	9
3	Construction of Star Types Using Ultrafilters	10
3.1	A Free Ultrafilter over the Naturals	10
3.2	Definition of <i>star</i> type constructor	10
3.3	Constructors for type ' <i>a star</i> '	11
3.4	Existence of a nonstandard natural number	11
3.5	Internal functions	12
3.6	Testing lifted booleans	12
3.7	Internal functions and predicates	12
3.8	Internal sets	13
3.9	Class constants	13
4	Transfer Principle	15
4.1	Preliminaries	16
4.2	Starting the transfer proof	16
4.3	Transfer introduction rules	17

4.3.1	Boolean operators	17
4.3.2	Equals, If	18
4.3.3	Quantifiers	18
4.3.4	Functions	18
4.3.5	Sets	18
4.3.6	Miscellaneous	19
4.4	Transfer tactic	20
5	Class Instances	21
5.1	HOL.thy	21
5.2	LOrder.thy	21
5.3	OrderedGroup.thy	21
5.4	Power.thy	23
5.5	Integ/Number.thy	23
6	Hyper-Natural numbers	24
6.0.1	Introduction properties	25
6.0.2	Elimination properties	25
6.0.3	Inductive (?) properties	25
6.1	Properties of "less than or equal"	26
6.2	Arithmetic operators	27
6.3	<i>min</i> and <i>max</i>	28
6.4	Basic rewrite rules for the arithmetic operators	28
6.5	Addition	28
6.6	Multiplication	29
6.7	Monotonicity of Addition	29
6.8	Additional theorems about "less than"	29
6.9	Difference	30
6.10	More results about difference	30
6.11	Monotonicity of Multiplication	32
7	Omega	33
7.1	An infinite natural number	33
7.2	Omega	33
7.3	Epsilon	34
8	Internal Sets	35
8.1	Properties of <i>Iset</i>	35
8.2	Properties of <i>Iset-of</i>	36
8.3	Finite internal sets	36



1 Zorn's Lemma

```
theory Zorn
imports Main
begin
```

The lemma and section numbers refer to an unpublished article [?].

```
constdefs
```

```
chain    :: 'a set set => 'a set set set
chain S == {F. F ⊆ S & (∀ x ∈ F. ∀ y ∈ F. x ⊆ y | y ⊆ x)}
```

```
super    :: ['a set set, 'a set set] => 'a set set set
super S c == {d. d ∈ chain S & c ⊂ d}
```

```
maxchain :: 'a set set => 'a set set set
maxchain S == {c. c ∈ chain S & super S c = {}}
```

```
succ     :: ['a set set, 'a set set] => 'a set set
succ S c ==
  if c ∉ chain S | c ∈ maxchain S
  then c else SOME c'. c' ∈ super S c
```

```
consts
```

```
TFin :: 'a set set => 'a set set set
```

```
inductive TFin S
```

```
  intros
```

```
    succI:      x ∈ TFin S ==> succ S x ∈ TFin S
```

```
    Pow-UnionI: Y ∈ Pow(TFin S) ==> Union(Y) ∈ TFin S
```

```
  monos      Pow-mono
```

1.1 Mathematical Preamble

```
lemma Union-lemma0: (∀ x ∈ C. x ⊆ A | B ⊆ x) ==> Union(C) ⊆ A | B ⊆ Union(C)
```

This is theorem *increasingD2* of ZF/Zorn.thy

```
lemma Abrial-axiom1: x ⊆ succ S x
```

```
lemmas TFin-UnionI = TFin.Pow-UnionI [OF PowI]
```

```
lemma TFin-induct:
```

```
  [| n ∈ TFin S;
    !!x. [| x ∈ TFin S; P(x) |] ==> P(succ S x);
    !!Y. [| Y ⊆ TFin S; Ball Y P |] ==> P(Union Y) |]
  ==> P(n)
```

```
lemma succ-trans: x ⊆ y ==> x ⊆ succ S y
```

Lemma 1 of section 3.1

lemma *TFin-linear-lemma1*:

$$\begin{aligned} & [[n \in TFin\ S; \ m \in TFin\ S; \\ & \quad \forall x \in TFin\ S. \ x \subseteq m \rightarrow x = m \mid succ\ S\ x \subseteq m \\ &]] \implies n \subseteq m \mid succ\ S\ m \subseteq n \end{aligned}$$

Lemma 2 of section 3.2

lemma *TFin-linear-lemma2*:

$$m \in TFin\ S \implies \forall n \in TFin\ S. \ n \subseteq m \rightarrow n = m \mid succ\ S\ n \subseteq m$$

Re-ordering the premises of Lemma 2

lemma *TFin-subsetD*:

$$[[n \subseteq m; \ m \in TFin\ S; \ n \in TFin\ S]] \implies n = m \mid succ\ S\ n \subseteq m$$

Consequences from section 3.3 – Property 3.2, the ordering is total

lemma *TFin-subset-linear*: $[[m \in TFin\ S; \ n \in TFin\ S]] \implies n \subseteq m \mid m \subseteq n$

Lemma 3 of section 3.3

lemma *eq-succ-upper*: $[[n \in TFin\ S; \ m \in TFin\ S; \ m = succ\ S\ m]] \implies n \subseteq m$

Property 3.3 of section 3.3

lemma *equal-succ-Union*: $m \in TFin\ S \implies (m = succ\ S\ m) = (m = Union(TFin\ S))$

1.2 Hausdorff's Theorem: Every Set Contains a Maximal Chain.

NB: We assume the partial ordering is \subseteq , the subset relation!

lemma *empty-set-mem-chain*: $(\{\} :: 'a\ set) \in chain\ S$

lemma *super-subset-chain*: $super\ S\ c \subseteq chain\ S$

lemma *maxchain-subset-chain*: $maxchain\ S \subseteq chain\ S$

lemma *mem-super-Ex*: $c \in chain\ S - maxchain\ S \implies \exists d. d \in super\ S\ c$

lemma *select-super*: $c \in chain\ S - maxchain\ S \implies$

$(@c'. c': \text{super } S c): \text{super } S c$

lemma *select-not-equals*: $c \in \text{chain } S - \text{maxchain } S \implies$
 $(@c'. c': \text{super } S c) \neq c$

lemma *succI3*: $c \in \text{chain } S - \text{maxchain } S \implies \text{succ } S c = (@c'. c': \text{super } S c)$

lemma *succ-not-equals*: $c \in \text{chain } S - \text{maxchain } S \implies \text{succ } S c \neq c$

lemma *TFin-chain-lemma4*: $c \in \text{TFin } S \implies (c :: 'a \text{ set set}): \text{chain } S$

theorem *Hausdorff*: $\exists c. (c :: 'a \text{ set set}): \text{maxchain } S$

1.3 Zorn's Lemma: If All Chains Have Upper Bounds Then There Is a Maximal Element

lemma *chain-extend*:

$[| c \in \text{chain } S; z \in S;$
 $\forall x \in c. x \leq (z :: 'a \text{ set}) |] \implies \{z\} \cup c \in \text{chain } S$

lemma *chain-Union-upper*: $[| c \in \text{chain } S; x \in c |] \implies x \subseteq \text{Union}(c)$

lemma *chain-ball-Union-upper*: $c \in \text{chain } S \implies \forall x \in c. x \subseteq \text{Union}(c)$

lemma *maxchain-Zorn*:

$[| c \in \text{maxchain } S; u \in S; \text{Union}(c) \subseteq u |] \implies \text{Union}(c) = u$

theorem *Zorn-Lemma*:

$\forall c \in \text{chain } S. \text{Union}(c): S \implies \exists y \in S. \forall z \in S. y \subseteq z \longrightarrow y = z$

1.4 Alternative version of Zorn's Lemma

lemma *Zorn-Lemma2*:

$\forall c \in \text{chain } S. \exists y \in S. \forall x \in c. x \subseteq y$
 $\implies \exists y \in S. \forall x \in S. (y :: 'a \text{ set}) \subseteq x \longrightarrow y = x$

Various other lemmas

lemma *chainD*: $[| c \in \text{chain } S; x \in c; y \in c |] \implies x \subseteq y \mid y \subseteq x$

lemma *chainD2*: $!(c :: 'a \text{ set set}). c \in \text{chain } S \implies c \subseteq S$

end

2 Filters and Ultrafilters

```
theory FreeUltrafilter
imports Zorn
begin
```

2.1 Definitions and basic properties

2.1.1 Filters

```
locale Filter =
  fixes F :: 'a set set
  assumes UNIV [iff]: UNIV ∈ F
  assumes empty [iff]: {} ∉ F
  assumes Int:      [[u ∈ F; v ∈ F]] ⇒ u ∩ v ∈ F
  assumes subset:  [[u ∈ F; u ⊆ v]] ⇒ v ∈ F
```

lemma (in *Filter*) *memD*: $A \in F \implies \neg A \notin F$

lemma (in *Filter*) *not-memI*: $\neg A \in F \implies A \notin F$

lemma (in *Filter*) *Int-iff*: $(x \cap y \in F) = (x \in F \wedge y \in F)$

2.1.2 Ultrafilters

```
locale Ultrafilter = Filter +
  assumes ultra: A ∈ F ∨ ¬ A ∈ F
```

lemma (in *Ultrafilter*) *memI*: $\neg A \notin F \implies A \in F$

lemma (in *Ultrafilter*) *not-memD*: $A \notin F \implies \neg A \in F$

lemma (in *Ultrafilter*) *not-mem-iff*: $(A \notin F) = (\neg A \in F)$

lemma (in *Ultrafilter*) *Compl-iff*: $(\neg A \in F) = (A \notin F)$

lemma (in *Ultrafilter*) *Un-iff*: $(x \cup y \in F) = (x \in F \vee y \in F)$

2.1.3 Free ultrafilters

```
locale FreeUltrafilter = Ultrafilter +
  assumes infinite: A ∈ F ⇒ infinite A
```

lemma (in *FreeUltrafilter*) *finite*: $\text{finite } A \implies A \notin F$

lemma (in *FreeUltrafilter*) *Filter*: *Filter* F

lemma (in *FreeUltrafilter*) *Ultrafilter*: *Ultrafilter* F

2.2 Maximal filter = ultrafilter

A filter F is an ultrafilter iff it is a maximal filter, i.e. whenever G is a filter and $F \subseteq G$ then $F = G$

Lemmas that shows existence of an extension to what was assumed to be a maximal filter. Will be used to derive contradiction in proof of property of ultrafilter.

lemma *extend-lemma1*: $UNIV \in F \implies A \in \{X. \exists f \in F. A \cap f \subseteq X\}$

lemma *extend-lemma2*: $F \subseteq \{X. \exists f \in F. A \cap f \subseteq X\}$

lemma (in *Filter*) *extend-Filter*:

assumes A : $- A \notin F$

shows *Filter* $\{X. \exists f \in F. A \cap f \subseteq X\}$ (is *Filter* ? X)

lemma (in *Filter*) *max-Filter-Ultrafilter*:

assumes *max*: $\bigwedge G. \llbracket \text{Filter } G; F \subseteq G \rrbracket \implies F = G$

shows *Ultrafilter-axioms* F

lemma (in *Ultrafilter*) *max-Filter*:

assumes G : *Filter* G and *sub*: $F \subseteq G$ **shows** $F = G$

2.3 Ultrafilter Theorem

A locale makes proof of Ultrafilter Theorem more modular

locale (open) *UFT* =

fixes *frechet* :: 'a set set

and *superfrechet* :: 'a set set set

assumes *infinite-UNIV*: *infinite* ($UNIV$:: 'a set)

defines *frechet-def*: $frechet \equiv \{A. finite (- A)\}$

and *superfrechet-def*: $superfrechet \equiv \{G. Filter\ G \wedge frechet \subseteq G\}$

lemma (in *UFT*) *superfrechetI*:

$\llbracket Filter\ G; frechet \subseteq G \rrbracket \implies G \in superfrechet$

lemma (in *UFT*) *superfrechetD1*:

$G \in superfrechet \implies Filter\ G$

lemma (in *UFT*) *superfrechetD2*:

$G \in superfrechet \implies frechet \subseteq G$

A few properties of free filters

lemma *Filter-cofinite*:

assumes *inf*: infinite (UNIV :: 'a set)
shows Filter {A:: 'a set. finite (- A)} (is Filter ?F)

We prove: 1. Existence of maximal filter i.e. ultrafilter; 2. Freeness property i.e ultrafilter is free. Use a locale to prove various lemmas and then export main result: The Ultrafilter Theorem

lemma (in UFT) *Filter-frechet*: Filter frechet

lemma (in UFT) *frechet-in-superfrechet*: frechet \in superfrechet

lemma (in UFT) *lemma-mem-chain-Filter*:
 $\llbracket c \in \text{chain superfrechet}; x \in c \rrbracket \implies \text{Filter } x$

2.3.1 Unions of chains of superfrechets

In this section we prove that superfrechet is closed with respect to unions of non-empty chains. We must show 1) Union of a chain is a filter, 2) Union of a chain contains frechet.

Number 2 is trivial, but 1 requires us to prove all the filter rules.

lemma (in UFT) *Union-chain-UNIV*:
 $\llbracket c \in \text{chain superfrechet}; c \neq \{\} \rrbracket \implies \text{UNIV} \in \bigcup c$

lemma (in UFT) *Union-chain-empty*:
 $c \in \text{chain superfrechet} \implies \{\} \notin \bigcup c$

lemma (in UFT) *Union-chain-Int*:
 $\llbracket c \in \text{chain superfrechet}; u \in \bigcup c; v \in \bigcup c \rrbracket \implies u \cap v \in \bigcup c$

lemma (in UFT) *Union-chain-subset*:
 $\llbracket c \in \text{chain superfrechet}; u \in \bigcup c; u \subseteq v \rrbracket \implies v \in \bigcup c$

lemma (in UFT) *Union-chain-Filter*:
assumes $c \in \text{chain superfrechet}$ **and** $c \neq \{\}$
shows Filter ($\bigcup c$)

lemma (in UFT) *lemma-mem-chain-frechet-subset*:
 $\llbracket c \in \text{chain superfrechet}; x \in c \rrbracket \implies \text{frechet} \subseteq x$

lemma (in UFT) *Union-chain-superfrechet*:
 $\llbracket c \neq \{\}; c \in \text{chain superfrechet} \rrbracket \implies \bigcup c \in \text{superfrechet}$

2.3.2 Existence of free ultrafilter

lemma (in UFT) *max-cofinite-Filter-Ex*:
 $\exists U \in \text{superfrechet}. \forall G \in \text{superfrechet}. U \subseteq G \implies U = G$

lemma (in *UFT*) *mem-superfrechet-all-infinite*:
 $\llbracket U \in \text{superfrechet}; A \in U \rrbracket \implies \text{infinite } A$

There exists a free ultrafilter on any infinite set

lemma (in *UFT*) *FreeUltrafilter-ex*:
 $\exists U :: 'a \text{ set set. FreeUltrafilter } U$

lemmas *FreeUltrafilter-Ex = UFT.FreeUltrafilter-ex*

end

3 Construction of Star Types Using Ultrafilters

theory *StarType*
imports *FreeUltrafilter*
begin

3.1 A Free Ultrafilter over the Naturals

constdefs
 $\text{FreeUltrafilterNat} :: \text{nat set set } (\mathcal{U})$
 $\mathcal{U} \equiv \text{SOME } U. \text{FreeUltrafilter } U$

lemma *FreeUltrafilter-FUFNat*: $\text{FreeUltrafilter } \mathcal{U}$

lemmas *Ultrafilter-FUFNat =*
 $\text{FreeUltrafilter-FUFNat } [\text{THEN } \text{FreeUltrafilter.Ultrafilter}]$

lemmas *Filter-FUFNat =*
 $\text{FreeUltrafilter-FUFNat } [\text{THEN } \text{FreeUltrafilter.Filter}]$

lemmas *FUFNat-empty [iff] =*
 $\text{Filter-FUFNat } [\text{THEN } \text{Filter.empty}]$

lemmas *FUFNat-UNIV [iff] =*
 $\text{Filter-FUFNat } [\text{THEN } \text{Filter.UNIV}]$

This rule takes the place of the old ultra tactic

lemma *ultra*:
 $\llbracket \{n. P n\} \in \mathcal{U}; \{n. P n \longrightarrow Q n\} \in \mathcal{U} \rrbracket \implies \{n. Q n\} \in \mathcal{U}$

3.2 Definition of *star* type constructor

constdefs

$starrel :: ((nat \Rightarrow 'a) \times (nat \Rightarrow 'a)) \text{ set}$
 $starrel \equiv \{(X, Y). \{n. X n = Y n\} \in \mathcal{U}\}$

typedef (Star) 'a star = (UNIV::(nat \Rightarrow 'a) set)//starrel

Proving that *starrel* is an equivalence relation

lemma *starrel-iff* [iff]: $((X, Y) \in starrel) = (\{n. X n = Y n\} \in \mathcal{U})$

lemma *equiv-starrel*: *equiv UNIV starrel*

lemmas *equiv-starrel-iff* =
eq-equiv-class-iff [OF *equiv-starrel UNIV-I UNIV-I*]
 — $(starrel \text{ `` } \{x\} = starrel \text{ `` } \{y\}) = ((x, y) \in starrel)$

lemma *starrel-in-Star*: $starrel \text{ `` } \{x\} \in Star$

lemma *eq-Abs-Star*:
 $(\bigwedge x. z = Abs-Star(starrel \text{ `` } \{x\}) \implies P) \implies P$

3.3 Constructors for type 'a star

constdefs

$star-n :: (nat \Rightarrow 'a) \Rightarrow 'a \text{ star}$ (**binder** STAR 10)
 $star-n X \equiv Abs-Star(starrel \text{ `` } \{X\})$

$star-of :: 'a \Rightarrow 'a \text{ star}$
 $star-of x \equiv STAR n. x$

theorem *star-cases*:
 $(\bigwedge X. x = star-n X \implies P) \implies P$

lemma *all-star-eq*: $(\forall x. P x) = (\forall X. P (star-n X))$

lemma *ex-star-eq*: $(\exists x. P x) = (\exists X. P (star-n X))$

lemma *star-n-eq-iff*: $(star-n X = star-n Y) = (\{n. X n = Y n\} \in \mathcal{U})$

lemma *star-of-inject*: $(star-of x = star-of y) = (x = y)$

3.4 Existence of a nonstandard natural number

constdefs

$nonstandard :: 'a \text{ star} \Rightarrow \text{bool}$
 $nonstandard x \equiv \forall a. x \neq star-of a$

lemma *nonstandardI*: $(\bigwedge a. x \neq star-of a) \implies nonstandard x$

lemma *nonstandardD*: $\text{nonstandard } x \implies x \neq \text{star-of } a$

lemma *ex-nonstandard-nat*: $\exists x::\text{nat } \text{star}. \text{nonstandard } x$

3.5 Internal functions

constdefs

Ifun :: $('a \Rightarrow 'b) \text{star} \Rightarrow 'a \text{star} \Rightarrow 'b \text{star}$

Ifun *f* $\equiv \lambda x. \text{Abs-Star}$

$(\bigcup F \in \text{Rep-Star } f. \bigcup X \in \text{Rep-Star } x. \text{starrel}''\{\lambda n. F n (X n)\})$

syntax *@Ifun* :: $('a \Rightarrow 'b) \text{star} \Rightarrow 'a \text{star} \Rightarrow 'b \text{star}$ (**infixl** \star 300)

translations $f \star x \rightleftharpoons \text{Ifun } f x$

lemma *Ifun-star-n*: $\text{star-n } F \star \text{star-n } X = (\text{STAR } n. F n (X n))$

lemma *Ifun [simp]*: $\text{star-of } f \star \text{star-of } x = \text{star-of } (f x)$

3.6 Testing lifted booleans

constdefs

Ibool :: $\text{bool star} \Rightarrow \text{bool}$

Ibool *b* $\equiv b = \text{star-of True}$

lemma *Ibool-star-n*: $\text{Ibool } (\text{star-n } P) = (\{n. P n\} \in \mathcal{U})$

lemma *Ibool [simp]*: $\text{Ibool } (\text{star-of } p) = p$

3.7 Internal functions and predicates

constdefs

Ifun-of :: $('a \Rightarrow 'b) \Rightarrow ('a \text{star} \Rightarrow 'b \text{star})$

Ifun-of *f* $\equiv \text{Ifun } (\text{star-of } f)$

Ifun2 :: $('a \Rightarrow 'b \Rightarrow 'c) \text{star} \Rightarrow ('a \text{star} \Rightarrow 'b \text{star} \Rightarrow 'c \text{star})$

Ifun2 *f* $\equiv \lambda x y. f \star x \star y$

Ifun2-of :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('a \text{star} \Rightarrow 'b \text{star} \Rightarrow 'c \text{star})$

Ifun2-of *f* $\equiv \lambda x y. \text{star-of } f \star x \star y$

Ipred :: $('a \Rightarrow \text{bool}) \text{star} \Rightarrow ('a \text{star} \Rightarrow \text{bool})$

Ipred *P* $\equiv \lambda x. \text{Ibool } (P \star x)$

Ipred-of :: $('a \Rightarrow \text{bool}) \Rightarrow ('a \text{star} \Rightarrow \text{bool})$

Ipred-of *P* $\equiv \lambda x. \text{Ibool } (\text{star-of } P \star x)$

Ipred2 :: ('a ⇒ 'b ⇒ bool) star ⇒ ('a star ⇒ 'b star ⇒ bool)
Ipred2 P ≡ λx y. Ibool (P ★ x ★ y)

Ipred2-of :: ('a ⇒ 'b ⇒ bool) ⇒ ('a star ⇒ 'b star ⇒ bool)
Ipred2-of P ≡ λx y. Ibool (star-of P ★ x ★ y)

lemma *Ifun-of [simp]*:
Ifun-of f (star-of x) = star-of (f x)

lemma *Ifun2-of [simp]*:
Ifun2-of f (star-of x) (star-of y) = star-of (f x y)

lemma *Ipred-of [simp]*:
Ipred-of P (star-of x) = P x

lemma *Ipred2-of [simp]*:
Ipred2-of P (star-of x) (star-of y) = P x y

lemmas *Ifun-defs* =
star-of-def *Ifun-of-def* *Ifun2-def* *Ifun2-of-def*
Ipred-def *Ipred-of-def* *Ipred2-def* *Ipred2-of-def*

3.8 Internal sets

constdefs
Iset :: 'a set star ⇒ 'a star set
Iset A ≡ {x. *Ipred2-of* (op ∈) x A}

Iset-of :: 'a set ⇒ 'a star set
Iset-of A ≡ *Iset* (star-of A)

lemma *Iset-star-n*:
(star-n X ∈ *Iset* (star-n A)) = ({n. X n ∈ A n} ∈ U)

3.9 Class constants

instance *star* :: (ord) ord
instance *star* :: (zero) zero
instance *star* :: (one) one
instance *star* :: (plus) plus
instance *star* :: (times) times
instance *star* :: (minus) minus
instance *star* :: (inverse) inverse
instance *star* :: (number) number
instance *star* :: (Divides.div) Divides.div
instance *star* :: (power) power
defs (overloaded)
star-zero-def: 0 ≡ star-of 0
star-one-def: 1 ≡ star-of 1

star-number-def: $\text{number-of } b \equiv \text{star-of } (\text{number-of } b)$
star-add-def: $(op +) \equiv \text{Ifun2-of } (op +)$
star-diff-def: $(op -) \equiv \text{Ifun2-of } (op -)$
star-minus-def: $\text{uminus} \equiv \text{Ifun-of } \text{uminus}$
star-mult-def: $(op *) \equiv \text{Ifun2-of } (op *)$
star-divide-def: $(op /) \equiv \text{Ifun2-of } (op /)$
star-inverse-def: $\text{inverse} \equiv \text{Ifun-of } \text{inverse}$
star-le-def: $(op \leq) \equiv \text{Ipred2-of } (op \leq)$
star-less-def: $(op <) \equiv \text{Ipred2-of } (op <)$
star-abs-def: $\text{abs} \equiv \text{Ifun-of } \text{abs}$
star-div-def: $(op \text{ div}) \equiv \text{Ifun2-of } (op \text{ div})$
star-mod-def: $(op \text{ mod}) \equiv \text{Ifun2-of } (op \text{ mod})$
star-power-def: $(op \wedge) \equiv \lambda x n. \text{Ifun2-of } (op \wedge) x (\text{star-of } n)$

lemmas *star-class-defs* =

star-zero-def *star-one-def* *star-number-def*
star-add-def *star-diff-def* *star-minus-def*
star-mult-def *star-divide-def* *star-inverse-def*
star-le-def *star-less-def* *star-abs-def*
star-div-def *star-mod-def* *star-power-def*

star-of preserves class operations

lemma *star-of-add*: $\text{star-of } (x + y) = \text{star-of } x + \text{star-of } y$

lemma *star-of-diff*: $\text{star-of } (x - y) = \text{star-of } x - \text{star-of } y$

lemma *star-of-minus*: $\text{star-of } (-x) = - \text{star-of } x$

lemma *star-of-mult*: $\text{star-of } (x * y) = \text{star-of } x * \text{star-of } y$

lemma *star-of-divide*: $\text{star-of } (x / y) = \text{star-of } x / \text{star-of } y$

lemma *star-of-inverse*: $\text{star-of } (\text{inverse } x) = \text{inverse } (\text{star-of } x)$

lemma *star-of-div*: $\text{star-of } (x \text{ div } y) = \text{star-of } x \text{ div } \text{star-of } y$

lemma *star-of-mod*: $\text{star-of } (x \text{ mod } y) = \text{star-of } x \text{ mod } \text{star-of } y$

lemma *star-of-power*: $\text{star-of } (x \wedge n) = \text{star-of } x \wedge n$

lemma *star-of-abs*: $\text{star-of } (\text{abs } x) = \text{abs } (\text{star-of } x)$

star-of preserves numerals

lemma *star-of-zero*: $\text{star-of } 0 = 0$

lemma *star-of-one*: $\text{star-of } 1 = 1$

lemma *star-of-number-of*: $\text{star-of } (\text{number-of } x) = \text{number-of } x$

star-of preserves orderings

lemma *star-of-less*: $(\text{star-of } x < \text{star-of } y) = (x < y)$

lemma *star-of-le*: $(\text{star-of } x \leq \text{star-of } y) = (x \leq y)$

lemma *star-of-eq*: $(\text{star-of } x = \text{star-of } y) = (x = y)$

As above, for 0

lemmas *star-of-0-less* = *star-of-less* [*of 0, simplified star-of-zero*]

lemmas *star-of-0-le* = *star-of-le* [*of 0, simplified star-of-zero*]

lemmas *star-of-0-eq* = *star-of-eq* [*of 0, simplified star-of-zero*]

lemmas *star-of-less-0* = *star-of-less* [*of - 0, simplified star-of-zero*]

lemmas *star-of-le-0* = *star-of-le* [*of - 0, simplified star-of-zero*]

lemmas *star-of-eq-0* = *star-of-eq* [*of - 0, simplified star-of-zero*]

As above, for 1

lemmas *star-of-1-less* = *star-of-less* [*of 1, simplified star-of-one*]

lemmas *star-of-1-le* = *star-of-le* [*of 1, simplified star-of-one*]

lemmas *star-of-1-eq* = *star-of-eq* [*of 1, simplified star-of-one*]

lemmas *star-of-less-1* = *star-of-less* [*of - 1, simplified star-of-one*]

lemmas *star-of-le-1* = *star-of-le* [*of - 1, simplified star-of-one*]

lemmas *star-of-eq-1* = *star-of-eq* [*of - 1, simplified star-of-one*]

lemmas *star-of-simps* =

star-of-add *star-of-diff* *star-of-minus*

star-of-mult *star-of-divide* *star-of-inverse*

star-of-div *star-of-mod*

star-of-power *star-of-abs*

star-of-zero *star-of-one* *star-of-number-of*

star-of-less *star-of-le* *star-of-eq*

star-of-0-less *star-of-0-le* *star-of-0-eq*

star-of-less-0 *star-of-le-0* *star-of-eq-0*

star-of-1-less *star-of-1-le* *star-of-1-eq*

star-of-less-1 *star-of-le-1* *star-of-eq-1*

declare *star-of-simps* [*simp*]

end

4 Transfer Principle

theory *Transfer*

imports *StarType*
begin

4.1 Preliminaries

These transform expressions like $\{n. f (P n)\} \in \mathcal{U}$

lemma *fuf-ex*:

$$(\{n. \exists x. P n x\} \in \mathcal{U}) = (\exists X. \{n. P n (X n)\} \in \mathcal{U})$$

lemma *fuf-not*: $(\{n. \neg P n\} \in \mathcal{U}) = (\{n. P n\} \notin \mathcal{U})$

lemma *fuf-conj*:

$$(\{n. P n \wedge Q n\} \in \mathcal{U}) = (\{n. P n\} \in \mathcal{U} \wedge \{n. Q n\} \in \mathcal{U})$$

lemma *fuf-disj*:

$$(\{n. P n \vee Q n\} \in \mathcal{U}) = (\{n. P n\} \in \mathcal{U} \vee \{n. Q n\} \in \mathcal{U})$$

lemma *fuf-imp*:

$$(\{n. P n \longrightarrow Q n\} \in \mathcal{U}) = (\{n. P n\} \in \mathcal{U} \longrightarrow \{n. Q n\} \in \mathcal{U})$$

lemma *fuf-iff*:

$$(\{n. P n = Q n\} \in \mathcal{U}) = ((\{n. P n\} \in \mathcal{U}) = (\{n. Q n\} \in \mathcal{U}))$$

lemma *fuf-all*:

$$(\{n. \forall x. P n x\} \in \mathcal{U}) = (\forall X. \{n. P n (X n)\} \in \mathcal{U})$$

lemma *fuf-if-bool*:

$$\begin{aligned} (\{n. \text{if } P n \text{ then } Q n \text{ else } R n\} \in \mathcal{U}) = \\ (\text{if } \{n. P n\} \in \mathcal{U} \text{ then } \{n. Q n\} \in \mathcal{U} \text{ else } \{n. R n\} \in \mathcal{U}) \end{aligned}$$

lemma *fuf-eq*:

$$(\{n. X n = Y n\} \in \mathcal{U}) = (\text{star-}n X = \text{star-}n Y)$$

lemma *fuf-if*:

$$\begin{aligned} (\text{STAR } n. \text{if } P n \text{ then } X n \text{ else } Y n) = \\ (\text{if } \{n. P n\} \in \mathcal{U} \text{ then } \text{star-}n X \text{ else } \text{star-}n Y) \end{aligned}$$

4.2 Starting the transfer proof

A transfer theorem asserts an equivalence $P \equiv P'$ between two related propositions. Proposition P may refer to constants having star types, like *'a star*. Proposition P' is syntactically similar, but only refers to ordinary types (i.e. P' is the un-starred version of P).

lemma *Trueprop-inject*: $(\text{Trueprop } P \equiv \text{Trueprop } Q) \equiv (P \equiv Q)$

This introduction rule starts each transfer proof.

lemma *transfer-start*:

$$P \equiv \{n. Q\} \in \mathcal{U} \implies \text{Trueprop } P \equiv \text{Trueprop } Q$$

4.3 Transfer introduction rules

The proof of a transfer theorem is completely syntax-directed. At each step in the proof, the top-level connective determines which introduction rule to apply. Each argument to the top-level connective generates a new subgoal.

Subgoals in a transfer proof always have the form of an equivalence. The left side can be any term, and may contain references to star types. The form of the right side depends on its type. At type *bool* it takes the form $\{n. P n\} \in \mathcal{U}$. At type '*a star*' it takes the form $STAR n. X n$. At type '*a star set*' it looks like $Iset (STAR n. A n)$. And at type '*a star* \Rightarrow '*b star*' it has the form $Ifun (STAR n. F n)$.

4.3.1 Boolean operators

lemma *transfer-not*:

$$\llbracket p \equiv \{n. P n\} \in \mathcal{U} \rrbracket \implies \neg p \equiv \{n. \neg P n\} \in \mathcal{U}$$

lemma *transfer-conj*:

$$\begin{aligned} \llbracket p \equiv \{n. P n\} \in \mathcal{U}; q \equiv \{n. Q n\} \in \mathcal{U} \rrbracket \\ \implies p \wedge q \equiv \{n. P n \wedge Q n\} \in \mathcal{U} \end{aligned}$$

lemma *transfer-disj*:

$$\begin{aligned} \llbracket p \equiv \{n. P n\} \in \mathcal{U}; q \equiv \{n. Q n\} \in \mathcal{U} \rrbracket \\ \implies p \vee q \equiv \{n. P n \vee Q n\} \in \mathcal{U} \end{aligned}$$

lemma *transfer-imp*:

$$\begin{aligned} \llbracket p \equiv \{n. P n\} \in \mathcal{U}; q \equiv \{n. Q n\} \in \mathcal{U} \rrbracket \\ \implies p \longrightarrow q \equiv \{n. P n \longrightarrow Q n\} \in \mathcal{U} \end{aligned}$$

lemma *transfer-iff*:

$$\begin{aligned} \llbracket p \equiv \{n. P n\} \in \mathcal{U}; q \equiv \{n. Q n\} \in \mathcal{U} \rrbracket \\ \implies p = q \equiv \{n. P n = Q n\} \in \mathcal{U} \end{aligned}$$

lemma *transfer-if-bool*:

$$\begin{aligned} \llbracket p \equiv \{n. P n\} \in \mathcal{U}; x \equiv \{n. X n\} \in \mathcal{U}; y \equiv \{n. Y n\} \in \mathcal{U} \rrbracket \\ \implies (\text{if } p \text{ then } x \text{ else } y) \equiv \{n. \text{if } P n \text{ then } X n \text{ else } Y n\} \in \mathcal{U} \end{aligned}$$

lemma *transfer-all-bool*:

$$\llbracket \bigwedge x. p x \equiv \{n. P n x\} \in \mathcal{U} \rrbracket \implies \forall x::\text{bool}. p x \equiv \{n. \forall x. P n x\} \in \mathcal{U}$$

lemma *transfer-ex-bool*:

$$\llbracket \bigwedge x. p x \equiv \{n. P n x\} \in \mathcal{U} \rrbracket \implies \exists x::\text{bool}. p x \equiv \{n. \exists x. P n x\} \in \mathcal{U}$$

4.3.2 Equals, If

lemma *transfer-star-eq*:

$$\llbracket x \equiv \text{star-}n X; y \equiv \text{star-}n Y \rrbracket \implies x = y \equiv \{n. X n = Y n\} \in \mathcal{U}$$

lemma *transfer-star-if*:

$$\begin{aligned} & \llbracket p \equiv \{n. P n\} \in \mathcal{U}; x \equiv \text{star-}n X; y \equiv \text{star-}n Y \rrbracket \\ & \implies (\text{if } p \text{ then } x \text{ else } y) \equiv \text{STAR } n. \text{ if } P n \text{ then } X n \text{ else } Y n \end{aligned}$$

4.3.3 Quantifiers

lemma *transfer-all*:

$$\begin{aligned} & \llbracket \bigwedge X. p (\text{star-}n X) \equiv \{n. P n (X n)\} \in \mathcal{U} \rrbracket \\ & \implies \forall x::'a \text{ star. } p x \equiv \{n. \forall x. P n x\} \in \mathcal{U} \end{aligned}$$

lemma *transfer-ex*:

$$\begin{aligned} & \llbracket \bigwedge X. p (\text{star-}n X) \equiv \{n. P n (X n)\} \in \mathcal{U} \rrbracket \\ & \implies \exists x::'a \text{ star. } p x \equiv \{n. \exists x. P n x\} \in \mathcal{U} \end{aligned}$$

lemma *transfer-ex1*:

$$\begin{aligned} & \llbracket \bigwedge X. p (\text{star-}n X) \equiv \{n. P n (X n)\} \in \mathcal{U} \rrbracket \\ & \implies \exists !x. p x \equiv \{n. \exists !x. P n x\} \in \mathcal{U} \end{aligned}$$

4.3.4 Functions

lemma *transfer-Ifun*:

$$\begin{aligned} & \llbracket f \equiv \text{star-}n F; x \equiv \text{star-}n X \rrbracket \\ & \implies f \star x \equiv \text{STAR } n. F n (X n) \end{aligned}$$

lemma *transfer-fun-eq*:

$$\begin{aligned} & \llbracket \bigwedge X. f (\text{star-}n X) = g (\text{star-}n X) \\ & \equiv \{n. F n (X n) = G n (X n)\} \in \mathcal{U} \rrbracket \\ & \implies f = g \equiv \{n. F n = G n\} \in \mathcal{U} \end{aligned}$$

4.3.5 Sets

lemma *transfer-Iset*:

$$\llbracket a \equiv \text{star-}n A \rrbracket \implies \text{Iset } a \equiv \text{Iset } (\text{STAR } n. A n)$$

lemma *transfer-Collect*:

$$\begin{aligned} & \llbracket \bigwedge X. p (\text{star-}n X) \equiv \{n. P n (X n)\} \in \mathcal{U} \rrbracket \\ & \implies \text{Collect } p \equiv \text{Iset } (\text{STAR } n. \text{Collect } (P n)) \end{aligned}$$

lemma *transfer-mem*:

$$\begin{aligned} & \llbracket x \equiv \text{star-}n X; a \equiv \text{Iset } (\text{star-}n A) \rrbracket \\ & \implies x \in a \equiv \{n. X n \in A n\} \in \mathcal{U} \end{aligned}$$

lemma *Iset-inject* [rule-format]:
 $\forall A B. (Iset A = Iset B) = (A = B)$

lemma *transfer-set-eq*:
 $\llbracket x \equiv Iset (star-n X); y \equiv Iset (star-n Y) \rrbracket$
 $\implies x = y \equiv \{n. X n = Y n\} \in \mathcal{U}$

4.3.6 Miscellaneous

lemma *transfer-Ibool*:
 $p \equiv star-n P \implies Ibool p \equiv \{n. P n\} \in \mathcal{U}$

lemma *transfer-STAR*: $STAR n. X n \equiv STAR n. X n$

lemma *transfer-bool*: $p \equiv \{n. p\} \in \mathcal{U}$

lemmas *transfer-intros* =
transfer-STAR
transfer-Ifun
transfer-fun-eq

transfer-not
transfer-conj
transfer-disj
transfer-imp
transfer-iff
transfer-if-bool
transfer-all-bool
transfer-ex-bool

transfer-all
transfer-ex
transfer-ex1

transfer-Ibool
transfer-bool
transfer-star-eq
transfer-star-if

transfer-set-eq
transfer-Iset
transfer-Collect
transfer-mem

4.4 Transfer tactic

We start by giving ML bindings for the theorems that will be used by the transfer tactic. Ideally, some of the lists of theorems should be expandable. To *star-class-defs* we would like to add theorems about *nat-of*, *int-of*, *meet*, *join*, etc. Also, we would like to create introduction rules for new constants.

Next we define the ML function *unstar-term*. This function takes a term, and gives back an equivalent term that does not contain any references to the *star* type constructor. Hopefully the resulting term will still be type-correct: Any constant whose type contains *star* should either be polymorphic (so that it will still work at the un-starred instance) or listed in *star-consts* (so it can be removed). Maybe *star-consts* should be extensible?

The *transfer-thm-of* function takes a term that represents a proposition, and proves that it is logically equivalent to the un-starred version. Assuming all goes well, it returns a theorem asserting the equivalence.

TODO: We need some error messages for when things go wrong. Errors may be caused by constants that don't have matching introduction rules, or quantifiers over wrong types.

Instead of applying the *transfer-start* rule right away, the proof of each transfer theorem starts with the transitivity rule $\llbracket P \equiv ?Q; ?Q \equiv P' \rrbracket \Longrightarrow P \equiv P'$, which introduces a new unbound schematic variable *?Q*. The first premise $P \equiv ?Q$ is solved by recursively using *transfer-start* and *transfer-intros*. Each rule application adds constraints to the form of *?Q*; by the time the first premise is proved, *?Q* is completely bound to the value of P' . Finally, the second premise is resolved with the reflexivity rule $P' \equiv P'$.

The delayed binding is necessary for the correct operation of the introduction rule *transfer-Ifun*: $\llbracket f \equiv \text{star-}n \ ?F; x \equiv \text{star-}n \ ?X \rrbracket \Longrightarrow f \star x \equiv \text{STAR } n. \ ?F \ n \ (?X \ n)$. With a subgoal of the form $f \star x \equiv \text{STAR } n. \ F \ n \ (X \ n)$, we would like to bind *?F* to *F* and *?X* to *X*. Unfortunately, higher-order unification finds more than one possible match, and binds *?F* to $\lambda x. x$ by default. If the right-hand side of the subgoal contains an unbound schematic variable instead of the explicit application $F \ n \ (X \ n)$, then we can ensure that there is only one possible way to match the rule.

Finally we set up the *transfer* method. It takes an optional list of definitions as arguments; they are passed along to *transfer-thm-of*, which expands the definitions before attempting to prove the transfer theorem.

Sample theorems

lemma *Ifun-inject*: $\bigwedge f \ g. (Ifun \ f = Ifun \ g) = (f = g)$

lemma *Ibool-inject*: $\bigwedge x y. (Ibool\ x = Ibool\ y) = (x = y)$

end

5 Class Instances

theory *StarClasses*
imports *Transfer*
begin

5.1 HOL.thy

instance *star* :: (*order*) *order*

instance *star* :: (*linorder*) *linorder*

5.2 LOrder.thy

Some extra trouble is necessary because the class axioms for *meet* and *join* use quantification over function spaces.

lemma *ex-star-fun*:
 $\exists f::('a \Rightarrow 'b)\ star. P\ (Ifun\ f)$
 $\implies \exists f::'a\ star \Rightarrow 'b\ star. P\ f$

lemma *ex-star-fun2*:
 $\exists f::('a \Rightarrow 'b \Rightarrow 'c)\ star. P\ (Ifun2\ f)$
 $\implies \exists f::'a\ star \Rightarrow 'b\ star \Rightarrow 'c\ star. P\ f$

instance *star* :: (*join-semilorder*) *join-semilorder*

instance *star* :: (*meet-semilorder*) *meet-semilorder*

instance *star* :: (*lorder*) *lorder*

lemma *star-join-def*: *join* \equiv *Ifun2-of join*

lemma *star-meet-def*: *meet* \equiv *Ifun2-of meet*

5.3 OrderedGroup.thy

instance *star* :: (*semigroup-add*) *semigroup-add*

instance *star* :: (*ab-semigroup-add*) *ab-semigroup-add*

instance *star* :: (*semigroup-mult*) *semigroup-mult*
instance *star* :: (*ab-semigroup-mult*) *ab-semigroup-mult*
instance *star* :: (*comm-monoid-add*) *comm-monoid-add*
instance *star* :: (*monoid-mult*) *monoid-mult*
instance *star* :: (*comm-monoid-mult*) *comm-monoid-mult*
instance *star* :: (*cancel-semigroup-add*) *cancel-semigroup-add*
instance *star* :: (*cancel-ab-semigroup-add*) *cancel-ab-semigroup-add*
instance *star* :: (*ab-group-add*) *ab-group-add*
instance *star* :: (*pordered-ab-semigroup-add*) *pordered-ab-semigroup-add*
instance *star* :: (*pordered-cancel-ab-semigroup-add*) *pordered-cancel-ab-semigroup-add*
instance *star* :: (*pordered-ab-semigroup-add-imp-le*) *pordered-ab-semigroup-add-imp-le*
instance *star* :: (*pordered-ab-group-add*) *pordered-ab-group-add*
instance *star* :: (*ordered-cancel-ab-semigroup-add*) *ordered-cancel-ab-semigroup-add*

instance *star* :: (*lordered-ab-group-meet*) *lordered-ab-group-meet*
instance *star* :: (*lordered-ab-group-meet*) *lordered-ab-group-meet*
instance *star* :: (*lordered-ab-group*) *lordered-ab-group*
instance *star* :: (*lordered-ab-group-abs*) *lordered-ab-group-abs*

Ring-and-Field.thy

instance *star* :: (*semiring*) *semiring*

instance *star* :: (*semiring-0*) *semiring-0*
instance *star* :: (*semiring-0-cancel*) *semiring-0-cancel*
instance *star* :: (*comm-semiring*) *comm-semiring*

instance *star* :: (*comm-semiring-0*) *comm-semiring-0*
instance *star* :: (*comm-semiring-0-cancel*) *comm-semiring-0-cancel*
instance *star* :: (*axclass-0-neq-1*) *axclass-0-neq-1*

instance *star* :: (*semiring-1*) *semiring-1*
instance *star* :: (*comm-semiring-1*) *comm-semiring-1*
instance *star* :: (*axclass-no-zero-divisors*) *axclass-no-zero-divisors*

instance *star* :: (*semiring-1-cancel*) *semiring-1-cancel*
instance *star* :: (*comm-semiring-1-cancel*) *comm-semiring-1-cancel*
instance *star* :: (*ring*) *ring*

```

instance star :: (comm-ring) comm-ring
instance star :: (ring-1) ring-1
instance star :: (comm-ring-1) comm-ring-1
instance star :: (idom) idom
instance star :: (field) field

instance star :: (division-by-zero) division-by-zero

instance star :: (pordered-semiring) pordered-semiring

instance star :: (pordered-cancel-semiring) pordered-cancel-semiring
instance star :: (ordered-semiring-strict) ordered-semiring-strict

instance star :: (pordered-comm-semiring) pordered-comm-semiring

instance star :: (pordered-cancel-comm-semiring) pordered-cancel-comm-semiring

instance star :: (ordered-comm-semiring-strict) ordered-comm-semiring-strict

instance star :: (pordered-ring) pordered-ring
instance star :: (lordered-ring) lordered-ring
instance star :: (axclass-abs-if) axclass-abs-if

instance star :: (ordered-ring-strict) ordered-ring-strict
instance star :: (pordered-comm-ring) pordered-comm-ring
instance star :: (ordered-semidom) ordered-semidom

instance star :: (ordered-idom) ordered-idom
instance star :: (ordered-field) ordered-field

```

5.4 Power.thy

Proving the class axiom *power-Suc* for type *'a star* is a little tricky, because it quantifies over values of type *nat*. The transfer principle does not handle quantification over non-star types in general, but we can work around this by fixing an arbitrary *nat* value, and then applying the transfer principle.

```

instance star :: (recpower) recpower

```

5.5 Integ/Number.thy

```

lemma star-of-nat-def: of-nat n  $\equiv$  star-of (of-nat n)

```

```

lemma int-diff-cases:

```

```

assumes prem:  $\bigwedge m n. z = \text{int } m - \text{int } n \implies P$  shows P

```

```

lemma star-of-int-def: of-int z  $\equiv$  star-of (of-int z)

```

```

instance star :: (number-ring) number-ring

lemma star-of-of-nat [simp]: star-of (of-nat n) = of-nat n

lemma star-of-of-int [simp]: star-of (of-int z) = of-int z

end

```

6 Hyper-Natural numbers

```

theory HypNat
imports StarClasses
begin

```

```

syntax hSuc :: nat star  $\Rightarrow$  nat star
translations hSuc  $\equiv$  Ifun-of Suc

```

```

lemma star-of-Suc: star-of (Suc n) = hSuc (star-of n)

```

Distinctness of constructors

```

lemma hSuc-not-Zero [iff]:  $\bigwedge m. hSuc\ m \neq 0$ 

```

```

lemma Zero-not-hSuc [iff]:  $\bigwedge m. 0 \neq hSuc\ m$ 

```

```

lemma hSuc-neq-Zero:  $\bigwedge m. hSuc\ m = 0 \implies R$ 

```

```

lemma Zero-neq-hSuc:  $\bigwedge m. 0 = hSuc\ m \implies R$ 

```

Injectiveness of *Suc*

```

lemma inj-def: inj f  $\equiv \forall x\ y. f\ x = f\ y \longrightarrow x = y$ 

```

```

lemma inj-hSuc: inj hSuc

```

```

lemma hSuc-inject:  $\bigwedge x\ y. hSuc\ x = hSuc\ y \implies x = y$ 

```

```

lemma hSuc-hSuc-eq [iff]:  $\bigwedge m\ n. (hSuc\ m = hSuc\ n) = (m = n)$ 

```

```

lemma hypnat-not-singleton:  $(\forall x. x = (0::nat\ star)) = False$ 

```

```

lemma n-not-hSuc-n:  $\bigwedge n. n \neq hSuc\ n$ 

```

```

lemma hSuc-n-not-n:  $\bigwedge t. hSuc\ t \neq t$ 

```

6.0.1 Introduction properties

lemma *star-lessI* [iff]: $\bigwedge n. n < hSuc\ n$

lemma *less-hSucI*: $\bigwedge i\ j. i < j \implies i < hSuc\ j$

lemma *zero-less-hSuc* [iff]: $\bigwedge n. 0 < hSuc\ n$

6.0.2 Elimination properties

lemma *star-lessE*:

$\bigwedge i\ k. \llbracket i < k; k = hSuc\ i \implies P; \bigwedge j. \llbracket i < j; k = hSuc\ j \rrbracket \implies P \rrbracket \implies P$

lemma *star-not-less0* [iff]: $\bigwedge n. \sim n < (0::nat\ star)$

lemma *star-less-zeroE*: $\bigwedge n. (n::nat\ star) < 0 \implies R$

lemma *less-hSucE*:

$\bigwedge m\ n. \llbracket m < hSuc\ n; m < n \implies P; m = n \implies P \rrbracket \implies P$

lemma *less-hSuc-eq*: $\bigwedge m\ n. (m < hSuc\ n) = (m < n \mid m = n)$

lemma *star-less-one* [iff]: $\bigwedge n. (n < (1::nat\ star)) = (n = 0)$

lemma *less-hSuc0* [iff]: $\bigwedge n. (n < hSuc\ 0) = (n = 0)$

lemma *hSuc-mono*: $\bigwedge m\ n. m < n \implies hSuc\ m < hSuc\ n$

”Less than” is antisymmetric, sort of

lemma *star-less-antisym*: $\bigwedge m\ n. \llbracket \neg n < m; n < hSuc\ m \rrbracket \implies m = n$

6.0.3 Inductive (?) properties

lemma *hSuc-lessI*: $\bigwedge m\ n. m < n \implies hSuc\ m \neq n \implies hSuc\ m < n$

lemma *hSuc-lessD*: $\bigwedge m\ n. hSuc\ m < n \implies m < n$

lemma *hSuc-lessE*:

$\bigwedge i\ k. \llbracket hSuc\ i < k; \bigwedge j. i < j \implies k = hSuc\ j \implies P \rrbracket \implies P$

lemma *hSuc-less-hSucD*: $\bigwedge m\ n. hSuc\ m < hSuc\ n \implies m < n$

lemma *hSuc-less-eq* [iff]: $\bigwedge m\ n. (hSuc\ m < hSuc\ n) = (m < n)$

lemma *less-trans-hSuc*: $\bigwedge i\ j\ k. i < j \implies j < k \implies hSuc\ i < k$

Can be used with *less-hSuc-eq* to get $n = m \vee n < m$
lemma *star-not-less-eq*: $\bigwedge m n. (\sim m < n) = (n < hSuc\ m)$

6.1 Properties of "less than or equal"

Was *le-eq-less-hSuc*, but this orientation is more useful

lemma *less-hSuc-eq-le*: $\bigwedge m n. (m < hSuc\ n) = (m \leq n)$

lemma *le-imp-less-hSuc*: $\bigwedge m n. m \leq n \implies m < hSuc\ n$

lemma *star-le0* [iff]: $\bigwedge n. (0::nat\ star) \leq n$

lemma *hSuc-n-not-le-n*: $\bigwedge n. \sim hSuc\ n \leq n$

lemma *star-le-0-eq* [iff]: $\bigwedge i. ((i::nat\ star) \leq 0) = (i = 0)$

lemma *le-hSuc-eq*: $\bigwedge m n. (m \leq hSuc\ n) = (m \leq n \mid m = hSuc\ n)$

lemma *le-hSucE*:

$\bigwedge m n. m \leq hSuc\ n \implies (m \leq n \implies R) \implies (m = hSuc\ n \implies R) \implies R$

lemma *star-leI*: $\bigwedge m n. \sim n < m \implies m \leq (n::nat\ star)$

lemma *star-leD*: $\bigwedge m n. m \leq n \implies \sim n < (m::nat\ star)$

lemmas *star-leE = star-leD* [elim-format]

lemma *hSuc-leI*: $\bigwedge m n. m < n \implies hSuc(m) \leq n$

lemma *hSuc-leD*: $\bigwedge m n. hSuc(m) \leq n \implies m \leq n$

Stronger version of *hSuc-leD*

lemma *hSuc-le-lessD*: $\bigwedge m n. hSuc\ m \leq n \implies m < n$

lemma *hSuc-le-eq*: $\bigwedge m n. (hSuc\ m \leq n) = (m < n)$

lemma *le-hSucI*: $\bigwedge m n. m \leq n \implies m \leq hSuc\ n$

For instance, $(hSuc\ m < hSuc\ n) = (hSuc\ m \leq n) = (m < n)$

lemmas *star-le-simps* = *order-less-imp-le less-hSuc-eq-le hSuc-le-eq*

Equivalence of $m \leq n$ and $m < n \vee m = n$

lemma *hSuc-le-mono* [iff]: $\bigwedge m\ n. (hSuc\ n \leq hSuc\ m) = (n \leq m)$

lemma *not-less-less-hSuc-eq*: $\bigwedge m\ n. \sim n < m ==> (n < hSuc\ m) = (n = m)$

lemma *le-less-hSuc-eq*: $\bigwedge m\ n. m \leq n ==> (n < hSuc\ m) = (n = m)$

lemmas *star-not-less-simps* = *not-less-less-hSuc-eq le-less-hSuc-eq*

6.2 Arithmetic operators

lemma *add-hSuc* [simp]: $\bigwedge m\ n. hSuc\ m + n = hSuc\ (m + n)$

lemma *star-diff-0* [simp]: $\bigwedge m. m - 0 = (m::nat\ star)$

lemma *mult-hSuc* [simp]: $\bigwedge m\ n. hSuc\ m * n = n + (m * n)$

lemma *not0-implies-hSuc*: $\bigwedge n. n \neq 0 ==> \exists m. n = hSuc\ m$

lemma *star-gr-implies-not0*: $!!m\ n::nat\ star. m < n ==> n \neq 0$

lemma *star-neq0-conv* [iff]: $!!n::nat\ star. (n \neq 0) = (0 < n)$

This theorem is useful with *blast*

lemma *star-gr0I*: $\bigwedge n. ((n::nat\ star) = 0 ==> False) ==> 0 < n$

lemma *gr0-conv-hSuc*: $\bigwedge n. (0 < n) = (\exists m. n = hSuc\ m)$

lemma *star-not-gr0* [iff]: $!!n::nat\ star. (\sim (0 < n)) = (n = 0)$

lemma *hSuc-le-D*: $\bigwedge n\ m'. (hSuc\ n \leq m') ==> (? m. m' = hSuc\ m)$

Useful in certain inductive arguments

lemma *less-hSuc-eq-0-disj*:
 $\bigwedge m n. (m < hSuc\ n) = (m = 0 \mid (\exists j. m = hSuc\ j \ \& \ j < n))$

6.3 *min* and *max*

lemma *star-min-0L* [*simp*]: $\bigwedge n. \min\ 0\ n = (0::nat\ star)$

lemma *star-min-0R* [*simp*]: $\bigwedge n. \min\ n\ 0 = (0::nat\ star)$

lemma *min-hSuc-hSuc* [*simp*]: $\bigwedge m n. \min\ (hSuc\ m)\ (hSuc\ n) = hSuc\ (\min\ m\ n)$

lemma *star-max-0L* [*simp*]: $\bigwedge n. \max\ 0\ n = (n::nat\ star)$

lemma *star-max-0R* [*simp*]: $\bigwedge n. \max\ n\ 0 = (n::nat\ star)$

lemma *max-hSuc-hSuc* [*simp*]: $\bigwedge m n. \max\ (hSuc\ m)\ (hSuc\ n) = hSuc\ (\max\ m\ n)$

6.4 Basic rewrite rules for the arithmetic operators

Difference

lemma *star-diff-0-eq-0* [*simp*]: $\bigwedge n. 0 - n = (0::nat\ star)$

lemma *diff-hSuc-hSuc* [*simp*]: $\bigwedge m n. hSuc\ m - hSuc\ n = m - n$

Could be (and is, below) generalized in various ways However, none of the generalizations are currently in the simpset, and I dread to think what happens if I put them in

lemma *hSuc-pred* [*simp*]: $\bigwedge n. 0 < n ==> hSuc\ (n - hSuc\ 0) = n$

6.5 Addition

lemma *add-hSuc-right* [*simp*]: $\bigwedge m n. m + hSuc\ n = hSuc\ (m + n)$

Reasoning about $m + 0 = 0$, etc.

lemma *star-add-is-0* [*iff*]: $!!m\ n::nat\ star. (m + n = 0) = (m = 0 \ \& \ n = 0)$

lemma *star-add-is-1*:

$\bigwedge m n. (m+n = hSuc\ 0) = (m = hSuc\ 0 \ \& \ n=0 \mid m=0 \ \& \ n = hSuc\ 0)$

lemma *star-one-is-add*:

$\bigwedge m n. (hSuc\ 0 = m + n) = (m = hSuc\ 0 \ \& \ n = 0 \mid m = 0 \ \& \ n = hSuc\ 0)$

lemma *star-add-gr-0* [*iff*]: $!!m\ n::nat\ star. (0 < m + n) = (0 < m \mid 0 < n)$

lemma *star-add-eq-self-zero*: $!!m n::nat\ star. m + n = m ==> n = 0$

6.6 Multiplication

lemma *mult-hSuc-right* [*simp*]: $\bigwedge m n. m * hSuc\ n = m + (m * n)$

lemma *star-mult-is-0* [*simp*]: $\bigwedge m n. ((m::nat\ star) * n = 0) = (m=0 \mid n=0)$

6.7 Monotonicity of Addition

lemma *less-imp-hSuc-add*: $\bigwedge m n. m < n ==> (\exists k. n = hSuc\ (m + k))$

6.8 Additional theorems about "less than"

A [clumsy] way of lifting $<$ monotonicity to \leq monotonicity

lemma *order-less-mono-imp-le-mono*:

assumes *lt-mono*: $!!i j::'a::order. i < j ==> f\ i < f\ j$

and *le*: $i \leq j$ **shows** $f\ i \leq (f\ j)::'b::order$

lemma *star-le-add2*: $\bigwedge m n. n \leq ((m + n)::nat\ star)$

lemma *star-le-add1*: $\bigwedge m n. n \leq ((n + m)::nat\ star)$

lemma *less-add-hSuc1*: $\bigwedge i m. i < hSuc\ (i + m)$

lemma *less-add-hSuc2*: $\bigwedge i m. i < hSuc\ (m + i)$

lemma *less-iff-hSuc-add*: $\bigwedge m n. (m < n) = (\exists k. n = hSuc\ (m + k))$

lemma *star-trans-le-add1*: $\bigwedge i j m. (i::nat\ star) \leq j ==> i \leq j + m$

lemma *star-trans-le-add2*: $\bigwedge i j m. (i::nat\ star) \leq j ==> i \leq m + j$

lemma *star-trans-less-add1*: $\bigwedge i j m. (i::nat\ star) < j ==> i < j + m$

lemma *star-trans-less-add2*: $\bigwedge i j m. (i::nat\ star) < j \implies i < m + j$

lemma *star-add-lessD1*: $\bigwedge i j k. i + j < (k::nat\ star) \implies i < k$

lemma *star-not-add-less1* [iff]: $\bigwedge i j. \sim (i + j < (i::nat\ star))$

lemma *star-not-add-less2* [iff]: $\bigwedge i j. \sim (j + i < (i::nat\ star))$

lemma *star-add-leD1*: $\bigwedge k m n. m + k \leq n \implies m \leq (n::nat\ star)$

lemma *star-add-leD2*: $\bigwedge k m n. m + k \leq n \implies k \leq (n::nat\ star)$

lemma *star-add-leE*:

$\bigwedge k m n. (m::nat\ star) + k \leq n \implies (m \leq n \implies k \leq n \implies R) \implies R$

needs !!k for *add-ac* to work

lemma *star-less-add-eq-less*:

$!!k\ l\ m\ n::nat\ star. k < l \implies m + l = k + n \implies m < n$

6.9 Difference

lemma *star-diff-self-eq-0* [simp]: $\bigwedge m. (m::nat\ star) - m = 0$

Addition is the inverse of subtraction: if $n \leq m$ then $n + (m - n) = m$.

lemma *star-add-diff-inverse*: $\bigwedge m n. \sim m < n \implies n + (m - n) = (m::nat\ star)$

lemma *star-le-add-diff-inverse* [simp]:

$\bigwedge m n. n \leq m \implies n + (m - n) = (m::nat\ star)$

lemma *star-le-add-diff-inverse2* [simp]:

$\bigwedge m n. n \leq m \implies (m - n) + n = (m::nat\ star)$

6.10 More results about difference

lemma *hSuc-diff-le*: $\bigwedge m n. n \leq m \implies hSuc\ m - n = hSuc\ (m - n)$

lemma *diff-less-hSuc*: $\bigwedge m n. m - n < hSuc\ m$

lemma *star-diff-le-self* [simp]: $\bigwedge m n. m - n \leq (m::nat\ star)$

lemma *star-less-imp-diff-less*: $\bigwedge j k n. (j::nat\ star) < k \implies j - n < k$

lemma *star-diff-diff-left*: $\bigwedge i j k. (i::nat\ star) - j - k = i - (j + k)$

lemma *hSuc-diff-diff* [simp]: $\bigwedge m n k. (hSuc\ m - n) - hSuc\ k = m - n - k$

lemma *diff-hSuc-less* [simp]: $\bigwedge n i. 0 < n \implies n - hSuc\ i < n$

This and the next few suggested by Florian Kammüller

lemma *star-diff-commute*: $\bigwedge i j k. (i::nat\ star) - j - k = i - k - j$

lemma *star-diff-add-assoc*:
 $\bigwedge i j k. k \leq (j::nat\ star) \implies (i + j) - k = i + (j - k)$

lemma *star-diff-add-assoc2*:
 $\bigwedge i j k. k \leq (j::nat\ star) \implies (j + i) - k = (j - k) + i$

lemma *star-diff-add-inverse*: $\bigwedge m n. (n + m) - n = (m::nat\ star)$

lemma *star-diff-add-inverse2*: $\bigwedge m n. (m + n) - n = (m::nat\ star)$

lemma *star-le-imp-diff-is-add*:
 $\bigwedge i j k. i \leq (j::nat\ star) \implies (j - i = k) = (j = k + i)$

lemma *star-diff-is-0-eq* [simp]: $\bigwedge m n. ((m::nat\ star) - n = 0) = (m \leq n)$

lemma *star-diff-is-0-eq'* [simp]: $\bigwedge m n. m \leq n \implies (m::nat\ star) - n = 0$

lemma *star-zero-less-diff* [simp]: $\bigwedge m n. (0 < n - (m::nat\ star)) = (m < n)$

lemma *star-less-imp-add-positive*:
 $\bigwedge i j. i < j \implies \exists k::nat\ star. 0 < k \ \& \ i + k = j$

lemma *star-diff-cancel*: $\bigwedge k m n. (k + m) - (k + n) = m - (n::nat\ star)$

lemma *star-diff-cancel2*: $\bigwedge k m n. (m + k) - (n + k) = m - (n::nat\ star)$

lemma *star-diff-add-0*: $\bigwedge m n. n - (n + m) = (0::nat\ star)$

Difference distributes over multiplication

lemma *star-diff-mult-distrib*:
 $\bigwedge k m n. ((m::nat\ star) - n) * k = (m * k) - (n * k)$

lemma *star-diff-mult-distrib2*:
 $\bigwedge k m n. k * ((m::nat\ star) - n) = (k * m) - (k * n)$

lemmas *star-nat-distrib* =
left-distrib right-distrib star-diff-mult-distrib star-diff-mult-distrib2

6.11 Monotonicity of Multiplication

lemma *star-mult-le-mono1*: $\bigwedge i j k. i \leq (j::\text{nat star}) \implies i * k \leq j * k$

lemma *star-mult-le-mono2*: $\bigwedge i j k. i \leq (j::\text{nat star}) \implies k * i \leq k * j$

\leq monotonicity, BOTH arguments

lemma *star-mult-le-mono*:

$\bigwedge i j k l. i \leq (j::\text{nat star}) \implies k \leq l \implies i * k \leq j * l$

lemma *star-mult-less-mono1*:

$\bigwedge i j k. (i::\text{nat star}) < j \implies 0 < k \implies i * k < j * k$

Differs from the standard *zero-less-mult-iff* in that there are no negative numbers.

lemma *star-nat-0-less-mult-iff* [simp]:

$\bigwedge m n. (0 < (m::\text{nat star}) * n) = (0 < m \ \& \ 0 < n)$

lemma *star-one-le-mult-iff* [simp]: $\bigwedge m n. (\text{hSuc } 0 \leq m * n) = (1 \leq m \ \& \ 1 \leq n)$

lemma *star-mult-eq-1-iff* [simp]: $\bigwedge m n. (m * n = \text{hSuc } 0) = (m = 1 \ \& \ n = 1)$

lemma *star-one-eq-mult-iff* [simp]: $\bigwedge m n. (\text{hSuc } 0 = m * n) = (m = 1 \ \& \ n = 1)$

lemma *star-mult-less-cancel2* [simp]:

$\bigwedge k m n. ((m::\text{nat star}) * k < n * k) = (0 < k \ \& \ m < n)$

lemma *star-mult-less-cancel1* [simp]:

$\bigwedge k m n. (k * (m::\text{nat star}) < k * n) = (0 < k \ \& \ m < n)$

lemma *star-mult-le-cancel1* [simp]:

$\bigwedge k m n. (k * (m::\text{nat star}) \leq k * n) = (0 < k \ \longrightarrow \ m \leq n)$

lemma *star-mult-le-cancel2* [simp]:

$\bigwedge k m n. ((m::\text{nat star}) * k \leq n * k) = (0 < k \ \longrightarrow \ m \leq n)$

lemma *star-mult-cancel2* [simp]:

$\bigwedge k m n. (m * k = n * k) = (m = n \ | \ (k = (0::\text{nat star})))$

lemma *star-mult-cancel1* [simp]:

$\bigwedge k m n. (k * m = k * n) = (m = n \ | \ (k = (0::\text{nat star})))$

lemma *hSuc-mult-less-cancel1*: $\bigwedge k m n. (\text{hSuc } k * m < \text{hSuc } k * n) = (m < n)$

lemma *hSuc-mult-le-cancel1*: $\bigwedge k m n. (\text{hSuc } k * m \leq \text{hSuc } k * n) = (m \leq n)$

lemma *hSuc-mult-cancel1*: $\bigwedge k m n. (hSuc\ k * m = hSuc\ k * n) = (m = n)$

Lemma for *gcd*

lemma *star-mult-eq-self-implies-10*:

$\bigwedge m n. (m::nat\ star) = m * n ==> n = 1 \mid m = 0$

end

7 Omega

theory *Omega*

imports *StarClasses HypNat*

begin

7.1 An infinite natural number

constdefs

infnat :: *nat star*

infnat \equiv *SOME* *x. nonstandard x*

lemma *nonstandard-infnat*: *nonstandard infnat*

lemma *infnat-neq-star-of*: *infnat \neq star-of n*

lemma *star-of-neq-infnat*: *star-of n \neq infnat*

lemma *infnat-neq-0* [*simp*]: *infnat \neq 0*

lemma *infnat-neq-1* [*simp*]: *infnat \neq 1*

lemma *less-nonstandard-nat*:

assumes *nonstandard k*

shows *star-of (n::nat) < k*

lemma *star-of-less-infnat*: *star-of n < infnat*

7.2 Omega

constdefs

omega :: '*a*::*ordered-semidom star*

omega \equiv *Ifun-of of-nat infnat*

syntax (*xsymbols*) *omega* :: '*a star* (ω)

syntax (*HTML output*) *omega* :: '*a star* (ω)

lemma *nonstandard-Ifun-of*:
assumes *inj: inj f and ns: nonstandard x*
shows *nonstandard (Ifun-of f x)*

lemma *less-Ifun-of-of-natI*:
 $\bigwedge n::nat\ star.\ star-of\ m < n$
 $\implies (of\ nat\ m::'a::ordered\ semidom\ star) < Ifun\ of\ of\ nat\ n$

lemma *of-nat-less-omega*: $of\ nat\ n < \omega$

lemma *zero-less-omega [simp]*: $0 < \omega$

lemma *one-less-omega [simp]*: $1 < \omega$

lemma *omega-neq-of-nat*: $\omega \neq of\ nat\ n$

lemma *omega-not-zero [simp]*: $\omega \neq 0$

lemma *omega-not-one [simp]*: $\omega \neq 1$

lemma *omega-not-mem-Nats*: $\omega \notin Nats$

lemma *nonstandard-omega*: *nonstandard* ω

lemma *omega-neq-star-of*: $\omega \neq star\ of\ n$

lemma *star-of-neq-omega*: $star\ of\ n \neq \omega$

7.3 Epsilon

constdefs

epsilon :: *'a::ordered-field star*
epsilon \equiv *inverse omega*

syntax (*xsymbols*) *epsilon* :: *'a star* (ε)

syntax (*HTML output*) *epsilon* :: *'a star* (ε)

lemma *inverse-epsilon*: *inverse* $\varepsilon = \omega$

lemma *epsilon-times-omega*: $\varepsilon * \omega = 1$

lemma *omega-times-epsilon*: $\omega * \varepsilon = 1$

lemma *zero-less-epsilon*: $0 < \varepsilon$

lemma *epsilon-neq-star-of*: $\varepsilon \neq star\ of\ x$

lemma *nonstandard-epsilon*: *nonstandard* ε

end

8 Internal Sets

theory *InternalSet*
imports *Omega*
begin

8.1 Properties of *Iset*

lemma *Iset-empty*: $Iset (star-of \{\}) = \{\}$

lemma *Iset-UNIV*: $Iset (star-of UNIV) = UNIV$

lemma *Iset-insert*:
 $\bigwedge a A. insert\ a\ (Iset\ A) = Iset\ (Ifun2-of\ insert\ a\ A)$

lemma *Iset-Un*:
 $\bigwedge A\ B. Iset\ A\ \cup\ Iset\ B = Iset\ (Ifun2-of\ (op\ \cup)\ A\ B)$

lemma *Iset-Int*:
 $\bigwedge A\ B. Iset\ A\ \cap\ Iset\ B = Iset\ (Ifun2-of\ (op\ \cap)\ A\ B)$

lemma *Iset-UNION*:
 $\bigwedge A\ B. (\bigcup_{x \in Iset\ A} Iset\ (B\ \star\ x)) = Iset\ (Ifun2-of\ UNION\ A\ B)$

lemma *Iset-INTER*:
 $\bigwedge A\ B. (\bigcap_{x \in Iset\ A} Iset\ (B\ \star\ x)) = Iset\ (Ifun2-of\ INTER\ A\ B)$

lemma *Iset-mem*:
 $\bigwedge x\ A. (x \in Iset\ A) = (Ipred2-of\ (op\ \in)\ x\ A)$

lemma *Iset-subset*:
 $\bigwedge A\ B. (Iset\ A \subseteq Iset\ B) = (Ipred2-of\ (op\ \subseteq)\ A\ B)$

lemma *Iset-diff*:
 $\bigwedge A\ B. (Iset\ A - Iset\ B) = Iset\ (Ifun2-of\ (op\ -)\ A\ B)$

lemma *Iset-Compl*:
 $\bigwedge A. (-\ Iset\ A) = Iset\ (Ifun-of\ uminus\ A)$

lemma *Iset-image*:
 $\bigwedge A\ f. Ifun\ f\ 'Iset\ A = Iset\ (Ifun2-of\ (op\ ')\ f\ A)$

8.2 Properties of *Iset-of*

lemma *Iset-of [simp]*:
 $(\text{star-of } x \in \text{Iset-of } A) = (x \in A)$

lemma *Iset-of-inject*: $(\text{Iset-of } A = \text{Iset-of } B) = (A = B)$

lemma *Iset-of-empty*: $\text{Iset-of } \{\} = \{\}$

lemma *Iset-of-UNIV*: $\text{Iset-of } \text{UNIV} = \text{UNIV}$

lemma *Iset-of-insert*:
 $\text{Iset-of } (\text{insert } x \ A) = \text{insert } (\text{star-of } x) (\text{Iset-of } A)$

lemma *Iset-of-Un*: $\text{Iset-of } (A \cup B) = \text{Iset-of } A \cup \text{Iset-of } B$

lemma *Iset-of-Int*: $\text{Iset-of } (A \cap B) = \text{Iset-of } A \cap \text{Iset-of } B$

lemma *Iset-of-diff*: $\text{Iset-of } (A - B) = \text{Iset-of } A - \text{Iset-of } B$

lemma *Iset-of-Compl*: $\text{Iset-of } (-A) = - \text{Iset-of } A$

lemma *Iset-of-subset*: $(\text{Iset-of } A \subseteq \text{Iset-of } B) = (A \subseteq B)$

lemma *Iset-of-UNION*:
 $\text{Iset-of } (\text{UNION } A \ B) = (\bigcup x \in \text{Iset-of } A. \text{Iset } (\text{Ifun-of } B \ x))$

lemma *Iset-of-INTER*:
 $\text{Iset-of } (\text{INTER } A \ B) = (\bigcap x \in \text{Iset-of } A. \text{Iset } (\text{Ifun-of } B \ x))$

lemma *Iset-of-image*:
 $\text{Iset-of } (f \ ' \ A) = \text{Ifun-of } f \ ' \ \text{Iset-of } A$

8.3 Finite internal sets

lemma *finite-Iset-of*: $\text{finite } A \implies \text{Iset-of } A = \text{star-of } \ ' \ A$

lemma *star-of-subset-Iset-of*: $\text{star-of } \ ' \ A \subseteq \text{Iset-of } A$

lemma $\text{Iset-of } A \cap \text{range } \text{star-of} = \text{star-of } \ ' \ A$

lemma *infinite-Iset-of*:
assumes *infinite* ($A :: 'a \ \text{set}$)
shows $\exists x \in \text{Iset-of } A. \text{nonstandard } x$

lemma $(\text{Iset-of } A = \text{star-of } \ ' \ A) = \text{finite } A$

instance *star* :: (*finite*) *finite*

end