Transfer Principle Proof Tactic for Nonstandard Analysis

Brian Huffman[*]

28th July 2005

Abstract

This paper describes a type constructor for nonstandard extensions of arbitrary types, generalizing the previous formalization of nonstandard analysis in Isabelle by Fleuriot and Paulson [1]. In addition to numeric types, nonstandard extensions of function spaces and set types also prove useful—they provide an elegant way to define internal functions and internal sets.

The paper also describes an implementation of the transfer principle of nonstandard analysis as an Isabelle proof tactic. When applied to an appropriate subgoal, the transfer tactic replaces the subgoal with a logically equivalent one that does not refer to nonstandard types. The tactic produces a proof of the equivalence; it does not generate any new axioms.

1 Introduction

Nonstandard analysis [2] was developed in the 1960s by Abraham Robinson to provide a rigorous foundation for the use of infinitesimal numbers in mathematics. Nonstandard analysis introduces an extension of the reals called the hyperreals, which includes infinitesimal and infinite numbers in addition to the standard reals. The hyperreals also preserve many properties of the reals, according to the transfer principle: Any true first order statement involving arithmetic on the reals may be reinterpreted as a true first order statement about the hyperreals.

Most calculus textbooks define derivatives and other notions of calculus using the epsilon-delta definition of limits. The hyperreal numbers require more work to formalize, but they offer significant advantages. First, using infinitesimals is often more intuitive: For example, in nonstandard analysis we can interpret the derivative $\frac{dy}{dx}$ as an actual ratio of two infinitesimal numbers. Second, by avoiding epsilon-delta limits, nonstandard analysis reduces the number of quantifiers we must deal with in proofs. This is especially important for automation in a theorem prover, where quantifiers are notoriously difficult to handle.

[*]OGI School of Science and Engineering at OHSU, Beaverton, OR 97006

1.1 Isabelle/HOL

Isabelle is a generic interactive theorem prover, which can be instantiated with various kinds of object-logics. Isabelle/HOL is an instantiation of higher order logic.

The formula syntax in Isabelle/HOL includes standard logical notation for connectives, quantifiers, and set operations. In addition, Isabelle has separate syntax for the meta-level logic: $\bigwedge$, $\Longrightarrow$, and $\equiv$ represent meta-level universal quantification, implication, and equality. There is also notation for nested meta-level implication: $[\![P_1; \ldots ; P_n]\!] \Longrightarrow R$ is short for $P_1 \Longrightarrow \cdots \Longrightarrow P_n \Longrightarrow R$. Other specialized Isabelle syntax will be introduced as it is used.

The syntax of types is similar to the language ML, except that Isabelle uses a double arrow ($\Rightarrow$) for function types. Some binary type constructors are written infix, as in the product type *nat* $\times$ *bool*; other type constructors are written postfix, as in *bool list* or *nat set*. Finally, *'a* and *'b* denote free type variables.

Isabelle theories declare new constants with the **consts** keyword. Definitions may be supplied later using **defs**; alternatively, constants may be declared and defined at once using **constdefs**. Theories introduce new types with the **typedef** command, which defines a type isomorphic to a given non-empty set. The keywords **lemma** and **theorem** introduce theorems.

1.2 HOL-Complex

Fleuriot and Paulson [1] have already developed a theory of nonstandard analysis in Isabelle/HOL: Their development comprises the HOL-Complex theory, which is part of the Isabelle distribution. The HOL-Complex theory has several parts. First, it includes a formalization of a few ordinary numeric types, including the rationals and the reals. Next, there is a formalization of free ultrafilters, which are used to define nonstandard types. Three separate nonstandard types are defined, each with their own specific set of operations: hyperreals, hypernaturals, and the hypercomplex numbers. Finally, these nonstandard types are used to formalize various concepts in real analysis.

The development described in the remainder of this paper may be considered as a potential replacement for the middle part of the HOL-Complex theory. It reuses much of the first part of HOL-Complex, including the formalization of standard numeric types and free ultrafilters. The remainder of the development consists of a framework for defining nonstandard types, defining operations on nonstandard types, and proving properties about them.

2 Type Constructor for Nonstandard Analysis

The theory starts with the definition of the *star* type constructor. The type *'a star* is defined as the set of equivalence classes of the *starrel* relation. (Note that double-slash // is Isabelle syntax for the set quotient operator.) In turn, *starrel* is defined in terms of an arbitrary free ultrafilter over the natural numbers, using Hilbert's indefinite choice operator. The development of nonstandard analysis is non-constructive, with the existence of a free ultrafilter proven using the axiom of choice. (See Appendix A for a review of free ultrafilters.)

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

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

typedef $'a\ \mathit{star} = (\mathit{UNIV} :: (\mathit{nat} \Rightarrow {'a})\ \mathit{set})\ //\ \mathit{starrel}$

The **typedef** command generates functions *Rep-star* and *Abs-star* to convert between the new type $'a\ \mathit{star}$ and the representation type $(\mathit{nat} \Rightarrow {'a})\ \mathit{set}$. Given a value of type $'a\ \mathit{star}$, *Rep-star* returns its equivalence class of sequences; *Abs-star* is the inverse mapping. The result is undefined when *Abs-star* is applied to a set that is not an equivalence class.

Abs-star is not very convenient to use by itself, so we now define some other basic functions to construct values of type $'a\ \mathit{star}$. The function *star-n* returns the value corresponding to the equivalence class of a given sequence. It is not injective, but it is surjective, which means that *star-n* may be used for doing case analysis on values of type $'a\ \mathit{star}$. The function *star-of* is an injective function that returns "standard" values of type $'a\ \mathit{star}$, which correspond to constant sequences.

constdefs
  $\mathit{star\text{-}n} :: (\mathit{nat} \Rightarrow {'a}) \Rightarrow {'a}\ \mathit{star}$
  $\mathit{star\text{-}n}\ X \equiv \mathit{Abs\text{-}star}\ \{Y.\ (X,Y) \in \mathit{starrel}\}$

  $\mathit{star\text{-}of} :: {'a} \Rightarrow {'a}\ \mathit{star}$
  $\mathit{star\text{-}of}\ x \equiv \mathit{star\text{-}n}\ (\lambda n.\ x)$

The HOL-Complex theory defines lifted versions of many functions on real numbers. For example, addition on reals (of type $\mathit{real} \Rightarrow \mathit{real} \Rightarrow \mathit{real}$) is lifted to addition on hyperreals (of type $\mathit{hypreal} \Rightarrow \mathit{hypreal} \Rightarrow \mathit{hypreal}$). The HOL-Complex theory defines a large number of these lifted functions, all in a similar way.

To generalize over this basic pattern of definition, we now define a function *Ifun* that is essentially a lifted version of the two-argument application function $(\lambda f\ x.\ f\ x)$. It is called *Ifun* because its range is exactly the set of internal functions. It is similar to the function *starfun-n* from HOL-Complex, but instead of taking a sequence of functions as an argument, it takes a value of type $({'a} \Rightarrow {'b})\ \mathit{star}$, which represents an equivalence class of such sequences.

constdefs
  $\mathit{Ifun} :: ({'a} \Rightarrow {'b})\ \mathit{star} \Rightarrow {'a}\ \mathit{star} \Rightarrow {'b}\ \mathit{star}\ (\mathit{infixl} \star\ 300)$
  $\mathit{Ifun}\ f \equiv \lambda x.\ \mathit{Abs\text{-}star}$
    $(\bigcup F{\in}\mathit{Rep\text{-}star}\ f.\ \bigcup X{\in}\mathit{Rep\text{-}star}\ x.\ \{Y.\ (\lambda n.\ F\ n\ (X\ n),\ Y) \in \mathit{starrel}\})$

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

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

Using *star-of* and *Ifun* as basic combinators, it is possible to define lifted versions of functions of any arity. It takes some non-trivial reasoning using the definition of *starrel* to establish the basic properties of *Ifun*, but we only have to do this once: For any function defined in terms of *Ifun*, we can then easily derive its properties from the *Ifun* lemmas.

One more useful combinator function is *unstar*, which converts from type *bool star* to type *bool*. It comes in very handy for defining lifted versions of predicates.

constdefs
$\mathit{unstar} :: \mathit{bool}\ \mathit{star} \Rightarrow \mathit{bool}$
$\mathit{unstar}\ b \equiv b = \mathit{star\text{-}of}\ \mathit{True}$

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

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

Next we use *star-of*, *Ifun*, and *unstar* to define several useful functions for defining lifted functions and predicates.

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

$\mathit{Ifun2} :: ('a \Rightarrow 'b \Rightarrow 'c)\ \mathit{star} \Rightarrow ('a\ \mathit{star} \Rightarrow 'b\ \mathit{star} \Rightarrow 'c\ \mathit{star})$
$\mathit{Ifun2}\ f \equiv \lambda x\ y.\ f \star x \star y$

$\mathit{Ifun2\text{-}of} :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('a\ \mathit{star} \Rightarrow 'b\ \mathit{star} \Rightarrow 'c\ \mathit{star})$
$\mathit{Ifun2\text{-}of}\ f \equiv \lambda x\ y.\ \mathit{star\text{-}of}\ f \star x \star y$

$\mathit{Ipred} :: ('a \Rightarrow \mathit{bool})\ \mathit{star} \Rightarrow ('a\ \mathit{star} \Rightarrow \mathit{bool})$
$\mathit{Ipred}\ P \equiv \lambda x.\ \mathit{unstar}\ (P \star x)$

$\mathit{Ipred\text{-}of} :: ('a \Rightarrow \mathit{bool}) \Rightarrow ('a\ \mathit{star} \Rightarrow \mathit{bool})$
$\mathit{Ipred\text{-}of}\ P \equiv \lambda x.\ \mathit{unstar}\ (\mathit{star\text{-}of}\ P \star x)$

$\mathit{Ipred2} :: ('a \Rightarrow 'b \Rightarrow \mathit{bool})\ \mathit{star} \Rightarrow ('a\ \mathit{star} \Rightarrow 'b\ \mathit{star} \Rightarrow \mathit{bool})$
$\mathit{Ipred2}\ P \equiv \lambda x\ y.\ \mathit{unstar}\ (P \star x \star y)$

$\mathit{Ipred2\text{-}of} :: ('a \Rightarrow 'b \Rightarrow \mathit{bool}) \Rightarrow ('a\ \mathit{star} \Rightarrow 'b\ \mathit{star} \Rightarrow \mathit{bool})$
$\mathit{Ipred2\text{-}of}\ P \equiv \lambda x\ y.\ \mathit{unstar}\ (\mathit{star\text{-}of}\ P \star x \star y)$

We can also use *star-of*, *Ifun*, and *unstar* to define a function that produces internal sets. It is similar to the function *starset-n* from HOL-Complex, but like *Ifun* it takes a value of type *'a set star* as an argument instead of a sequence.

constdefs
$\mathit{Iset} :: 'a\ \mathit{set}\ \mathit{star} \Rightarrow 'a\ \mathit{star}\ \mathit{set}$
$\mathit{Iset}\ A \equiv \{x.\ \mathit{Ipred2\text{-}of}\ (\mathit{op} \in)\ x\ A\}$

$\mathit{Iset\text{-}of} :: 'a\ \mathit{set} \Rightarrow 'a\ \mathit{star}\ \mathit{set}$
$\mathit{Iset\text{-}of}\ A \equiv \mathit{Iset}\ (\mathit{star\text{-}of}\ A)$

lemma *Iset-star-n*:
$(\mathit{star\text{-}n}\ X \in \mathit{Iset}\ (\mathit{star\text{-}n}\ A)) = (\{n.\ X\ n \in A\ n\} \in \mathcal{U})$

Finally, we can define many standard overloaded constants for the *star* type constructor, using the lifting functions defined above.

defs (**overloaded**)
star-zero-def: $0 \equiv \textit{star-of } 0$
star-one-def: $1 \equiv \textit{star-of } 1$
star-number-def: $\textit{number-of } b \equiv \textit{star-of } (\textit{number-of } b)$
star-add-def: $(\textit{op} +) \equiv \textit{Ifun2-of } (\textit{op} +)$
star-diff-def: $(\textit{op} -) \equiv \textit{Ifun2-of } (\textit{op} -)$
star-minus-def: $\textit{uminus} \equiv \textit{Ifun-of uminus}$
star-mult-def: $(\textit{op} *) \equiv \textit{Ifun2-of } (\textit{op} *)$
star-divide-def: $(\textit{op} /) \equiv \textit{Ifun2-of } (\textit{op} /)$
star-inverse-def: $\textit{inverse} \equiv \textit{Ifun-of inverse}$
star-le-def: $(\textit{op} \leq) \equiv \textit{Ipred2-of } (\textit{op} \leq)$
star-less-def: $(\textit{op} <) \equiv \textit{Ipred2-of } (\textit{op} <)$
star-abs-def: $\textit{abs} \equiv \textit{Ifun-of abs}$
star-div-def: $(\textit{op div}) \equiv \textit{Ifun2-of } (\textit{op div})$
star-mod-def: $(\textit{op mod}) \equiv \textit{Ifun2-of } (\textit{op mod})$
star-power-def: $(\textit{op} \hat{}) \equiv \lambda x\ n.\ \textit{Ifun-of } (\lambda x.\ x \hat{}\ n)\ x$

3 Transfer Tactic

The transfer principle is a meta-mathematical theorem, which says that many propositions over nonstandard types are logically equivalent to syntactically similar propositions over standard types. Such a principle cannot be encoded as a single theorem in Isabelle because it quantifies over valid theorems, which are not first class entities in Isabelle's object logic. Instead, we can encode it as an algorithm that can generate any of an entire family of theorems.

In this development, the transfer principle is implemented as a proof tactic. When applied to an appropriate subgoal, it replaces the subgoal with a logically equivalent one that does not refer to the *star* type constructor. The tactic produces a proof of the equivalence; it does not generate any new axioms.

3.1 Stating the Equivalence

When the transfer tactic is first called, it obtains a syntactic representation of the current subgoal as an ML term datatype. The term is expected to represent a proposition about *star* types. The job of the tactic is to replace the subgoal with an equivalent one that does not mention *star* types.

Therefore, the first thing that the tactic does is to traverse the term, removing all references to the *star* type constructor from the types of constants, and also completely removing any constants like *Ifun*, *star-of*, *Iset*, etc. Hopefully the resulting term will still be type-correct: The types of constants like *Ifun* and *star-of* reduce to the type of an identity function when un-starred, so they can safely be removed. Any other constant whose type mentions *star* should be polymorphic, so that it will still work at the un-starred instance. For example, the equality operator is valid at both type $'a\ \textit{star} \Rightarrow\ 'a\ \textit{star} \Rightarrow \textit{bool}$ and type $'a \Rightarrow\ 'a \Rightarrow \textit{bool}$.

Next, the tactic creates a new term that states an equivalence between the original term and its un-starred version. If this equivalence term type checks, then the tactic will attempt to prove the equivalence.

3.2 Starting the Equivalence Proof

We start with an equivalence between terms of type *prop*, which is the type of truth values in Isabelle's meta-logic. The first step is to reduce this to an equivalence between terms of type *bool*, the type of truth values in the object logic. Therefore we unfold a set of rewrite rules that convert all of the meta-level quantification, implication, and equalities to ordinary object-level constructs.

The next step involves unfolding a few different sets of definitions. First of all, the tactic unfolds all of the definitions for the overloaded constants listed in the previous section, including *star-zero-def*, *star-add-def*, *star-le-def*, etc. The tactic also unfolds the definitions of constants like *Ifun-of*, *Ipred*, and *star-of*. After all this unfolding, the remaining equivalence subgoal should only mention the constants *Ifun*, *Iset*, *star-n*, boolean operators, quantifiers, and a few other constants that the transfer principle knows about.

Once the equivalence has been reduced to a manageable form, the following introduction rule starts the remainder of the proof. (*Trueprop* is the function in Isabelle that maps from type *bool* to type *prop*—it is usually implicit.)

lemma *transfer-start*:
$P \equiv \{n.\ Q\} \in \mathcal{U} \Longrightarrow \mathit{Trueprop}\ P \equiv \mathit{Trueprop}\ Q$

3.3 Transfer Introduction Rules

The remaining steps of the proof are completely syntax-directed. At each step, the top-level connective determines an appropriate introduction rule to apply. Each argument to the top-level connective generates a new subgoal, each of which also takes the form of an equivalence.

The transfer introduction rules for unary negation, conjunction, and existential quantification over *star* types are all proven using the properties of free ultrafilters. Similar introduction rules for other boolean operators and quantifiers can be derived from these rules.

lemma *transfer-not*:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U}]\!] \Longrightarrow \neg\ p \equiv \{n.\ \neg\ P\ n\} \in \mathcal{U}$

lemma *transfer-conj*:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ q \equiv \{n.\ Q\ n\} \in \mathcal{U}]\!]$
$\Longrightarrow p \wedge q \equiv \{n.\ P\ n \wedge Q\ n\} \in \mathcal{U}$

lemma *transfer-ex*:
$[\![\bigwedge X.\ p\ (\mathit{star\text{-}n}\ X) \equiv \{n.\ P\ n\ (X\ n)\} \in \mathcal{U}]\!]$
$\Longrightarrow \exists\, x{::}'a\ \mathit{star}.\ p\ x \equiv \{n.\ \exists\, x.\ P\ n\ x\} \in \mathcal{U}$

The above introduction rules only deal with equivalences between booleans. However, it is usually the case that some subterms will have non-boolean types, for example *star* types or sets. Rules for constants with these other types are similar, but have a different form on the right hand side: For *star* types the right side contains an application of *star-n*, and for sets the right hand side starts with *Iset*.

lemma *transfer-eq*:
$[\![x \equiv \mathit{star\text{-}n}\ X;\ y \equiv \mathit{star\text{-}n}\ Y]\!] \Longrightarrow x = y \equiv \{n.\ X\ n = Y\ n\} \in \mathcal{U}$

lemma *transfer-if*:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ x \equiv \textit{star-n}\ X;\ y \equiv \textit{star-n}\ Y]\!]$
$\Longrightarrow (\textit{if}\ p\ \textit{then}\ x\ \textit{else}\ y) \equiv \textit{star-n}\ (\lambda n.\ \textit{if}\ P\ n\ \textit{then}\ X\ n\ \textit{else}\ Y\ n)$

lemma *transfer-mem*:
$[\![x \equiv \textit{star-n}\ X;\ a \equiv \textit{Iset}\ (\textit{star-n}\ A)]\!]$
$\Longrightarrow x \in a \equiv \{n.\ X\ n \in A\ n\} \in \mathcal{U}$

lemma *transfer-set-eq*:
$[\![a \equiv \textit{Iset}\ (\textit{star-n}\ A);\ b \equiv \textit{Iset}\ (\textit{star-n}\ B)]\!]$
$\Longrightarrow a = b \equiv \{n.\ A\ n = B\ n\} \in \mathcal{U}$

Each application of one of these introduction rules results in a smaller subgoal. Eventually, the left hand side of the subgoals will reduce to an atomic term, which can be discharged by one of the following rules.

lemma *transfer-star-n*: $\textit{star-n}\ X \equiv \textit{star-n}\ (\lambda n.\ X\ n)$

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

After the equivalence proof is done, the transfer tactic uses the resulting equivalence theorem as a rewrite rule to replace the old subgoal with the new, un-starred version.

3.4 Using the Transfer Tactic

I have used the transfer tactic to convert two large collections of theorems from ordinary types to star types. First, it is easily proved that the *star* type constructor inherits membership in a large number of axiomatic type classes from its argument type; for example, for any ordered field *'a*, *'a star* is also an ordered field. For each class axiom, invoking the transfer tactic reduces the proof obligation at type *'a star* to type *'a*, which may be discharged by applying the class axiom at that type.

One axiomatic type class that presents a slight difficulty is the *recpower* class, whose axioms assert that the exponentiation operator ($\textit{op}\ \hat{}\ ::\ 'a \Rightarrow \textit{nat} \Rightarrow\ 'a$) is defined in a standard way. When we try to prove that *'a star* inherits the *recpower* class from *'a*, we get the following two subgoals:

$\bigwedge(a{::}'a\ \textit{star}).\ a\ \hat{}\ 0 = 1$
$\bigwedge(a{::}'a\ \textit{star})\ (n{::}\textit{nat}).\ a\ \hat{}\ \textit{Suc}\ n = a * a\ \hat{}\ n$

The transfer tactic handles the first subgoal just fine, because it only uses universal quantification over star types. Indeed, this is the situation for the vast majority of numeric class axioms. However, the second subgoal also quantifies over type *nat*, which the transfer tactic does not handle. The workaround is to prove, as a lemma, a modified version of the second subgoal where only *a* is universally quantified, and *n* is a free variable.

The second major use case is the conversion of the entire Isabelle/HOL natural number theory into the *nat star* type. For each theorem in the original theory file, a new version is stated that uses type *nat star* in place of *nat*, and *Ifun-of Suc* in place of *Suc*. It is also necessary to add explicit meta-universal quantifiers for each free variable of type *nat star* in each theorem, because the transfer tactic is not allowed to change the types of free variables in the middle of a proof. (This does not affect the resulting theorem, because Isabelle automatically discharges meta-universal quantifiers after the end of any proof.) The proof script for each of the theorems is a one-liner: First apply the transfer tactic, and then apply the original rule from the theory of natural numbers.

The only theorems from the natural number theory that require extra modifications are the induction rules, because they involve universal quantification over predicates $P :: \mathit{nat} \Rightarrow \mathit{bool}$. In other presentations of nonstandard analysis, the transfer principle produces induction rules for *nat star* with extra side conditions that restrict $P :: \mathit{nat\ star} \Rightarrow \mathit{bool}$ to be an internal predicate. In this development, however, no side conditions are necessary: The transfer tactic can generate induction rules that quantify over $P :: (\mathit{nat} \Rightarrow \mathit{bool})\ \mathit{star}$, whose type can represent only internal predicates.

4 Conclusion

The development described in this paper borrows much from the HOL-Complex theory. My definition of a type constructor for nonstandard types is a straightforward generalization of the definitions of nonstandard types in the HOL-Complex theory. I have also reused its formalization of free ultrafilters, and its formalization of the real number system.

One original contribution of this work is to refactor the common patterns of definitions into a few basic combinators, from which many other concepts may be defined. Together with the transfer principle, this allows reasoning about nonstandard types without ever having to look at how they are represented in terms of sequences using ultrafilters. To the end user, it is almost as if nonstandard analysis had been formalized axiomatically.

The primary benefit to this work, however, is the degree of automation that it brings to the formalization of nonstandard analysis. Many theorems that previously required complex proofs can now be done in a straightforward and almost trivial manner.

References

[1] Jacques D. Fleuriot and Lawrence C. Paulson. Mechanizing nonstandard real analysis. In *LMS J. Comput. Math.*, pages 140–190, 2000.

[2] Abraham Robinson. *Non-standard Analysis*. Princeton Landmarks in Mathematics. Princeton University Press, 1996.

A Free Ultrafilters

This section is a quick overview of free ultrafilters and how they are used to define the hyperreal numbers. Let U be a set of sets of naturals, and let the elements of U be called the "large" sets. Here are some properties of large sets that we might like to have:

1. The set of all naturals is large, and the empty set is not.
2. If A and B are large, then so is $A \cap B$.

3. If A is large and $A \subseteq B$, then B is large.

4. If A is not large, then the complement of A is large.

5. If A is finite, then A is not large.

If U satisfies the first three properties, then we say that U is a *filter*. If U additionally satisfies Property 4, then U is an *ultrafilter*; and if U satisfies all five then U is a *free ultrafilter*. Using Zorn's Lemma (an equivalent form of the axiom of choice) it is possible to prove the existence of a free ultrafilter over the natural numbers.

We can use a free ultrafilter to define the hyperreal numbers as equivalence classes of sequences of real numbers. We will say that two sequences X and Y are equivalent ($X \sim Y$) if their agreement set $\{n.\ X(n) = Y(n)\}$ is large. Properties 1–3 ensure that this is an equivalence relation.

We can also use the ultrafilter to define a less-than relation on sequences of reals. We will say that $X < Y$ if the set $\{n.\ X(n) < Y(n)\}$ is large. Properties 1–3 ensure that the less-than relation respects the equivalence relation on sequences. Property 4 ensures that the order trichotomy of reals is preserved: For any sequences X and Y, exactly one of $X < Y$, $Y < X$, or $X \sim Y$ must hold.

Given any real number x, we can define a corresponding "standard" hyperreal as the equivalence class of the constant sequence $(x, x, x, \ldots)$. Property 1 ensures that this is an injective mapping. Property 5 is used to show that this mapping is not surjective. Consider the sequence $(1, 2, 3, \ldots)$ which does not contain more than one occurrence of any number. This sequence is not equivalent to any constant sequence, since the agreement set between the two is finite, and thus not large. In fact, the hyperreal corresponding to the sequence $(1, 2, 3, \ldots)$ is greater than any standard number.