Set Theory as a Computational Logic: I. From Foundations to Functions[*]

Lawrence C. Paulson
Computer Laboratory
University of Cambridge

4 November 1992

Abstract

Zermelo-Fraenkel (ZF) set theory is widely regarded as unsuitable for automated reasoning. But a computational logic has been formally derived from the ZF axioms using Isabelle. The library of theorems and derived rules, with Isabelle's proof tools, support a natural style of proof. The paper describes the derivation of rules for descriptions, relations and functions, and discusses interactive proofs of Cantor's Theorem, the Composition of Homomorphisms challenge [3], and Ramsey's Theorem [2].

[*]Research funded by SERC grant GR/G53279 and by the ESPRIT Basic Research Action 3245 'Logical Frameworks'. Isabelle has enjoyed long-standing support from the British SERC, dating from the Alvey Programme (grant GR/E0355.7).

Contents

1 Introduction

A great many formalisms have been proposed for reasoning about computer systems: Hoare logics, modal/temporal logics, constructive logics, etc. Some are designed to handle the needs of a specialized problem domain. Others are designed for ease of implementation; consider the highly successful Boyer/Moore logic [4], which consists of a quantifier-free first-order logic augmented with carefully chosen principles of recursion.

Specialized logics often have limited expressive power. Since there is no clear dividing line between computational reasoning and arbitrary mathematics, perhaps we should adopt a fully general mathematical formalism — one that has no difficulty with concepts such as infinite objects, equivalence classes or sets of functions. A general formalism may seem impossible to implement effectively, the more so if it must compete with other logics in their specialized domains. But M. J. C. Gordon's work demonstrates that higher-order logic, which is a general formalism, can be applied successfully to specialized domains such as hardware verification [6, 7].

Axiomatic set theory is older and more general than higher-order logic. Can it be used for verification? Set theory is commonly regarded as unworkable. Yet Noël [10], Quaife [18] and Saaltink [20], in their different ways, show that complex set theory proofs are possible. Such work seems to require great care and effort; the next step is to make the proof process easy.

The drawbacks of set theory are well known. It is extremely low-level, with strange definitions like $\langle a, b\rangle \equiv \{\{a\}, \{a, b\}\}$ and $3 \equiv \{0, 1, 2\}$; and since it has no form of information hiding, it admits strange theorems like $\{a\} \in \langle a, b\rangle$ and $2 \in 3$. But to compensate, set theory has tremendous expressive power. Its basic concepts are few and are widely understood.

Finally, set theory has no type checking — every object is a set — while higher-order logic has infinitely many types. This is the clearest difference between the two formalisms. Whether type checking is bad or good is perhaps a matter of taste, just as it is with programming languages.

The paper proceeds as follows. The next two sections introduce Isabelle and axiomatic set theory. Further sections sketch the Isabelle development of basic concepts such as relations and functions. Next come interactive proofs of three small examples: ordered pairing, Cantor's Theorem, and the Composition of Homomorphisms challenge [3]. Ramsey's Theorem, a more realistic example, permits a comparison between Isabelle and other theorem provers [2]. The remaining sections discuss related work and draw conclusions.

2 Isabelle

The formalization and proofs discussed below have been done with the help of Isabelle, an interactive theorem prover [14]. Isabelle is **generic**: it supports a range of formalisms, including modal, first-order, higher-order, and intuitionistic logics. Isabelle's generic capabilities are vital for using set theory, whose axioms are far too low-level for most proofs. Isabelle's version of set theory includes derived theories of relations, functions and type constructions, which can be regarded as logics in their own right. These theories exploit Isabelle's treatment of syntax, variable-binding operators, and derived rules.

Isabelle works directly with schematic inference rules of the form

$$[\![\phi_1; \ldots; \phi_n]\!] \Longrightarrow \phi.$$

Rules are combined by a generalization of Horn clause resolution. Theorems are proved not by refutation, but in the affirmative. Joining rules by resolution constructs a proof tree, whose root is the conclusion.

Such rules are theorems of Isabelle's meta-logic, which is a fragment of higher-order logic. The symbol $\Longrightarrow$ is meta-implication; the notation $[\![\phi_1; \ldots; \phi_n]\!] \Longrightarrow \phi$ abbreviates

$$\phi_1 \Longrightarrow (\cdots \Longrightarrow (\phi_n \Longrightarrow \phi) \cdots).$$

The symbol $\bigwedge$ is another meta-connective, a universal quantifier, for expressing generality in rules. The symbol $\equiv$, which is meta-equality, expresses definitions. Elsewhere [12] I discuss how to formalize object-logics in the meta-logic, and how to prove that the formalization is correct.

Expressions in the meta-logic are typed λ-terms, and λ-abstraction handles an object-logic's quantifiers and other variable-binding operators. The presence of λ-terms means that Isabelle cannot use ordinary unification. **Higher-order unification** is undecidable in the general case but works well in practice — particularly for enforcing quantifier rule provisos of the form 'x not free in ...' [12].

For backward proof, a rule of the form $[\![\phi_1; \ldots; \phi_n]\!] \Longrightarrow \phi$ can represent a **proof state**; the ultimate goal is ϕ and the subgoals still unsolved are $\phi_1, \ldots, \phi_n$. An **initial** proof state has the form $\phi \Longrightarrow \phi$, with one subgoal, and a **final** proof state has the form ϕ, with no subgoals. The final proof state *is itself* the desired theorem.

Tactics are functions that transform proof states. A backward proof proceeds by applying tactics in succession to the initial state, reaching a final state. The tactic `resolve_tac` performs Isabelle's form of Horn clause resolution; it attempts to unify the conclusion of some inference rule with a subgoal, replacing it by the rule's instantiated premises. This is proof checking.

Isabelle also supports automated reasoning. Each tactic maps a proof state to a **lazy list** of possible next states. Backtracking is therefore possible and tactics can implement search strategies such as depth-first, best-first and iterative deepening. **Tacticals** are operators for combining tactics. They typically express control structures, ranging from basic sequencing to search strategies. Isabelle provides several powerful, generic tools:

- The **classical reasoner** applies naive heuristics to prove theorems in the style of the sequent calculus. Despite its naiveity, it can prove many nontrivial theorems, including nearly all of Pelletier's graded problems short of Schubert's Steamroller [17]. As an interactive tool it is valuable. It is not restricted to first-order logic, but exploits any natural deduction rules. It can prove several key lemmas for Ramsey's Theorem [2].

- The **simplifier** applies rewrite rules to a goal, then attempts to prove the rewritten goal using a user-supplied tactic. A conditional rewrite rule is applied only if recursive simplification proves the instantiated condition. Contextual information is also used, rewriting $x = t \rightarrow \psi(x)$ to $x = t \rightarrow \psi(t)$. Rewriting works not just for equality, but for any reflexive/transitive relation enjoying congruence laws. Used with the classical reasoner, it can prove Boyer et al.'s challenge problem, that the composition of homomorphisms is a homomorphism [3].

Isabelle does not find proofs automatically. Proofs require a skilled user, who must decide which lemmas to prove and which tools to apply. Each tool must be given a set of appropriate lemmas. For instance, the proof about composition of homomorphisms requires lemmas about the composition of functions. Sometimes no tool is appropriate and we must use proof checking; even these proofs can be concise if they exploit derived rules and tacticals.

3 Set theory

Axiomatic set theory was developed in response to paradoxes such as Russell's. Sets could not be arbitrary collections of the form $\{x \,.\, \phi(x)\}$, pulled out of a hat. They had to be constructed, starting from a few given sets. Operations for constructing new sets included union, powerset and replacement.

Replacement is the most powerful set constructor. If A is a set, and the binary predicate $\phi(x, y)$ is single-valued for all x in A, then replacement yields the image of A under the predicate ϕ. Formally, if

$$\forall_{x \in A} \,.\, \forall y\, z \,.\, \phi(x,y) \land \phi(x,z) \rightarrow y = z$$

then there exists a set $\mathcal{R}(A, \phi)$ such that

$$b \in \mathcal{R}(A,\phi) \leftrightarrow (\exists_{x \in A} \,.\, \phi(x,b)).$$

Replacement entails the principle of **Separation**. Let A be a set and $\psi(x)$ a unary predicate. Separation yields a set, written $\{x \in A \,.\, \psi(x)\}$, consisting of those elements of A that satisfy ψ:

$$a \in \{x \in A \,.\, \psi(x)\} \leftrightarrow a \in A \land \psi(a)$$

A **class** is an arbitrary collection of sets. Elements of the class $\{x \,.\, \psi(x)\}$ are not restricted to elements of some other set. Every set B is a class, namely $\{x \,.\, x \in B\}$. Many classes are too big to be sets, such as the **universal class**, $V \equiv \{x \,.\, x = x\}$. If V were a set then we could obtain Russell's Paradox via Separation: define the set $R \equiv \{x \in V \,.\, x \notin x\}$, then $R \in R \leftrightarrow R \notin R$. We could define R as a class, namely $R \equiv \{x \,.\, x \notin x\}$, but this yields no paradox because a proper class cannot be a member of another class: $R \in R$ is false.

3.1 Which axiom system?

The two main axiom systems for set theory, Zermelo-Fraenkel (ZF) and von Neumann-Bernays-Gödel (NBG), differ in their treatment of classes. In ZF, variables range over sets; classes do not exist at all, but we may regard unary predicates as classes if we like. In NBG, variables range over classes, and $A \in V$ expresses that the class A is actually a set. The two axiom systems are similar in strength. Most set theorists prefer ZF because they are interested in sets, not classes. Moreover, NBG is tiresome to use — it frequently requires showing that certain classes are sets. Even $a \in \{a\}$ holds only if $a \in V$; see Lemma 9 of Boyer et al. [3].

ZF has one serious drawback: Replacement is expressed by an axiom scheme, parametrized by the predicate ϕ. A textbook application of the Compactness Theorem demonstrates that ZF can have no finite axiom system in first-order logic. Thus, it is unsuitable for first-order resolution theorem provers. Boyer et al. [3] advocate NBG because it is finite. Quaife [18] has simplified their clauses for NBG and proved several hundred results, using the resolution prover Otter.

Isabelle can express axiom schemes, since its meta-logic is higher-order. In the axiom of Replacement, the binary predicate ϕ is a variable of type $[i,i] \Rightarrow o$, which is the type of functions that map two individuals to a truth value. Separation can be derived in its schematic form, where the unary predicate ψ is a variable of type $i \Rightarrow o$.

Schemes can nonetheless cause problems with search. The goal

$$t \in ?A,$$

could be refined, instantiating the unknown $?A$ (a 'logical variable'), to the subgoal

$$t \in \{x \in ?B \,.\, ?\psi(x)\}.$$

This can be refined, by the principle of Separation, to the two subgoals

$$t \in ?B \qquad \text{and} \qquad ?\psi(t).$$

The subgoal $t \in ?B$ can be refined exactly like $t \in ?A$, making the search loop; the subgoal $?\psi(t)$ is totally unconstrained, since it consists of a formula unknown. Had we proved $t \in ?A$ by instantiating $?A$ to $\{t\}$, we might have invalidated other goals involving $?A$. Such a situation arises in the proof of Cantor's Theorem (see §5.2). Automatic tools seldom cope; the user can help by explicitly instantiating unknowns such as $?A$.

3.2 The Zermelo-Fraenkel axioms in Isabelle

The ZF axioms from Suppes [22, page 238] are expressed using Isabelle's formulation of classical first-order logic. For clarity, the exposition uses standard mathematical notation rather than Isabelle's ASCII substitutes [16]. We begin by defining the bounded quantifiers:

$$\begin{aligned} \forall_{x\in A} \,.\, P(x) &\equiv \forall x \,.\, x \in A \to P(x) \\ \exists_{x\in A} \,.\, P(x) &\equiv \exists x \,.\, x \in A \land P(x) \end{aligned}$$

Taking membership ($\in$) as a primitive binary relation, we define the subset relation:

$$A \subseteq B \quad\equiv\quad \forall_{x\in A} \,.\, x \in B$$

The following axioms are standard:

$$\begin{array}{rcll}
A = B & \leftrightarrow & A \subseteq B \land B \subseteq A & \text{(Extensionality)}\\
A \in \bigcup(C) & \leftrightarrow & (\exists_{B\in C} \,.\, A \in B) & \text{(Union)}\\
A \in \wp(B) & \leftrightarrow & A \subseteq B & \text{(Powerset)}\\
A = \emptyset & \lor & (\exists_{x\in A} \,.\, \forall_{y\in x} \,.\, y \notin A) & \text{(Foundation)}
\end{array}$$

Replacement is expressed by a rule whose premise asserts that ϕ is single-valued:

$$\frac{\forall_{x\in A} \,.\, \forall y\, z \,.\, \phi(x,y) \land \phi(x,z) \to y = z}{b \in \mathcal{R}(A,\phi) \leftrightarrow (\exists_{x\in A} \,.\, \phi(x,b))} \qquad \text{(Replacement)}$$

These are all the axioms apart from Infinity, which is not discussed in this paper, and Choice, which I have not used at all.

3.3 Natural deduction rules for set theory

The theory above is largely in the form of logical equivalences; perhaps we could develop a transformational calculus. But for general purposes, I prefer to derive natural deduction rules. Here are some examples.

From the definition $A \subseteq B \equiv \forall_{x\in A}.x \in B$ we obtain introduction and elimination rules for $\subseteq$:

$$\frac{\begin{array}{c}[x \in A]_x\\ \vdots\\ x \in B\end{array}}{A \subseteq B}\,(\subseteq I) \qquad \frac{A \subseteq B \quad c \in A}{c \in B}\,(\subseteq E)$$

Rule $(\subseteq I)$ discharges the assumption $x \in A$; it holds provided x is not free in the conclusion or other assumptions. Here and below, premises indicate such provisos by subscripting the affected variable.

From the Union axiom, we may derive introduction and elimination rules for $\bigcup$:

$$\frac{B \in C \quad A \in B}{A \in \bigcup(C)}\,(\bigcup I) \qquad \frac{A \in \bigcup(C) \qquad \begin{array}{c}[A \in X \quad X \in C]_X\\ \vdots\\ \theta\end{array}}{\theta}\,(\bigcup E)$$

Rule $(\bigcup E)$ discharges two assumptions, and has another 'not free' proviso on X.

Natural deduction rules break down formulae one level at a time. They are more readable than sequent rules because they leave the context implicit: each rule mentions only the assumptions it discharges. Forward and backward reasoning can be intermixed. Isabelle can use natural deduction rules to support purely backward reasoning, in the style of the sequent calculus. The resulting automated proof procedures resemble those based on semantic tableaux.

3.4 A simplified form of Replacement

The Axiom of Replacement, as traditionally expressed, is awkward for natural deduction. The introduction and elimination rules for $b \in \mathcal{R}(A, \phi)$ both require an additional premise stating that ϕ is single-valued. Defining a new form of Replacement reduces this proof burden. If $\phi(x, y)$ is a binary predicate, then let

$$\phi'(x, y) \quad \equiv \quad (\exists! z \, . \, \phi(x, z)) \wedge \phi(x, y).$$

Since $\exists! z \, . \, \phi(x, z)$ means there exists a *unique* z such that $\phi(x, z)$, the definition ensures that $\phi'(x, y)$ is single-valued.[1] Moreover, if $\phi(x, y)$ is already single-valued then the two predicates are equivalent. We define the new form of Replacement (with a nice notation) by

$$\{y \, . \, x \in A, \phi(x, y)\} \quad \equiv \quad \mathcal{R}(A, \phi')$$

and easily obtain the equivalence

$$b \in \{y \, . \, x \in A, \phi(x, y)\} \leftrightarrow (\exists_{x \in A} \, . \, \phi(x, b) \wedge (\forall y \, . \, \phi(x, y) \to y = b)).$$

This equivalence is unconditional. It never asks whether $\phi(x, y)$ is single-valued for all x in A, only for some value of x such that $\phi(x, b)$.

Using the new definition, we derive natural deduction rules. The introduction rule includes a simplified premise about the single-valued property. The elimination rule requires no such premise; on the contrary, it discharges an assumption involving this property. (The assumption, omitted below for clarity, is $\forall y \, . \, \phi(x, y) \to y = b$.)

$$\frac{a \in A \quad \phi(a, b) \quad \begin{array}{c} [\phi(a, y)]_y \\ \vdots \\ y = b \end{array}}{b \in \{y \, . \, x \in A, \phi(x, y)\}} \, (\mathcal{R}I) \qquad \frac{b \in \{y \, . \, x \in A, \phi(x, y)\} \quad \begin{array}{c} [x \in A \quad \phi(x, b)]_x \\ \vdots \\ \theta \end{array}}{\theta} \, (\mathcal{R}E)$$

3.5 Functional Replacement

Suppose that f is a unary operator on sets — not a set-theoretic function, which is a set of pairs, but a meta-level function such as $\wp$ or $\bigcup$. Since the predicate $\phi(x, y) \equiv (y = f(x))$ is obviously single-valued, define

$$\{f(x) \, . \, x \in A\} \quad \equiv \quad \{y \, . \, x \in A, y = f(x)\}.$$

This form of Replacement illustrates why single-valued predicates are sometimes called **class functions**. Isabelle can express meta-level functions by abstraction in its typed λ-calculus.

Functional replacement, with the basic $\bigcup$ operator, expresses a more familiar form of union:

$$\bigcup_{x \in A} B(x) \quad \equiv \quad \bigcup(\{B(x) \, . \, x \in A\})$$

[1]Isabelle expresses ϕ' in terms of ϕ using meta-level λ-abstraction.

The corresponding natural deduction rules are

$$\frac{a \in A \quad b \in B(a)}{b \in (\bigcup_{x \in A} . B(x))}\,(\bigcup \mathcal{R}I) \qquad \frac{b \in (\bigcup_{x \in A} . B(x)) \qquad \begin{matrix}[x \in A \quad b \in B(x)]_x \\ \vdots \\ \theta\end{matrix}}{\theta}\,(\bigcup \mathcal{R}E)$$

3.6 Separation

Given a set A and a unary predicate ψ, Separation yields a set consisting of those elements of A that satisfy ψ. Separation is easily defined in terms of Replacement:

$$\{x \in A \,.\, \psi(x)\} \quad \equiv \quad \{y \,.\, x \in A, x = y \wedge \psi(x)\}$$

The natural deduction rules have simple derivations:

$$\frac{a \in A \quad \psi(a)}{a \in \{x \in A \,.\, \psi(x)\}} \qquad \frac{a \in \{x \in A \,.\, \psi(x)\}}{a \in A} \qquad \frac{a \in \{x \in A \,.\, \psi(x)\}}{\psi(a)}$$

Using Separation, we can define general intersection:

$$\bigcap(C) \quad \equiv \quad \{x \in \bigcup(C) \,.\, \forall_{Y \in C} \,.\, x \in Y\}$$

The empty intersection, $\bigcap(\emptyset)$, causes difficulties. It would like to contain everything, but there is no universal set; $\bigcap(\emptyset)$ should be undefined. But Isabelle's set theory does not formalize the notion of definedness; all terms are defined. Because $\bigcup(\emptyset) = \emptyset$, we obtain the perverse (but harmless) result $\bigcap(\emptyset) = \emptyset$.

4 Deriving a theory of functions

The next developments are tightly linked. We define unordered pairs, then binary unions and intersections, and obtain finite sets of arbitrary size. Then we can define descriptions and ordered pairs. Finally, we can define Cartesian products, binary relations and functions. The resulting theory includes a sort of λ-calculus with Π and Σ types. All the proofs have been done in Isabelle.

4.1 Finite sets and the boolean operators

Unordered pairing is frequently taken as primitive, but it can be defined in terms of Replacement [22, page 237]. Observe that $\wp(\wp(\emptyset))$ contains two distinct elements, $\emptyset$ and $\wp(\emptyset)$.

$$\texttt{Upair}(a, b) \quad \equiv \quad \{y \,.\, x \in \wp(\wp(\emptyset)),\ (x = \emptyset \wedge y = a) \vee (x = \wp(\emptyset) \wedge y = b)\}$$

Tedious but elementary reasoning yields the key property:

$$c \in \texttt{Upair}(a, b) \leftrightarrow (c = a \vee c = b).$$

Now we can define binary union, intersection and (while we are at it) set difference:

$$\begin{aligned} A \cup B &\equiv \bigcup(\mathtt{Upair}(A, B)) \\ A \cap B &\equiv \bigcap(\mathtt{Upair}(A, B)) \\ A - B &\equiv \{x \in A \,.\, x \notin B\} \end{aligned}$$

Finite sets are traditionally obtained as binary unions of unordered pairs. Isabelle's treatment is inspired by Lisp. Define

$$\mathtt{cons}(a, B) \equiv \mathtt{Upair}(a, a) \cup B.$$

Thus $\mathtt{cons}(a, B)$ augments B with the element a; we obtain

$$c \in \mathtt{cons}(a, B) \leftrightarrow (c = a \vee c \in B).$$

In Isabelle, the notation $\{a_1, \ldots, a_n\}$ expands to $\mathtt{cons}(a_1, \ldots, \mathtt{cons}(a_n, \emptyset) \ldots)$.

4.2 Descriptions

Compared with Suppes [22], Isabelle's axioms take one liberty. They do not merely assert the existence of powersets, unions and replacements, but give them names: $\wp(A)$, $\bigcup(A)$ and $\mathcal{R}(A, \phi)$. There is nothing wrong with assigning notation to objects, provided they are unique, and Suppes does so informally.

By introducing these names, we gain the power to define a general description operator:

$$\iota x \,.\, \psi(x) \equiv \bigcup\{y \,.\, x \in \{\emptyset\}, \psi(y)\}$$

Observe the peculiar usage of Replacement. The formula $\psi(y)$ is single-valued in x and y simply because x is restricted to a singleton set. If there exists a unique a satisfying $\psi(a)$, then $\iota x \,.\, \psi(x)$ equals a. (If not then it equals $\emptyset$, although this fact matters little.)

Because it demands uniqueness, $\iota x \,.\, \psi(x)$ is much weaker than Hilbert's description $\epsilon x \,.\, \psi(x)$, which embodies a strong version of the Axiom of Choice. Unique descriptions are still useful, as we shall see; their properties are summed up by two derived rules:

$$\frac{\psi(a) \qquad \begin{matrix}[\psi(x)]_x \\ \vdots \\ x = a\end{matrix}}{(\iota x \,.\, \psi(x)) = a}\ (\iota{=}) \qquad \frac{\exists! x \,.\, \psi(x)}{\psi(\iota x \,.\, \psi(x))}\ (\iota I)$$

4.3 Ordered pairs

The definition $\langle a, b\rangle \equiv \{\{a\}, \{a, b\}\}$ is perhaps the most famous (or notorious) fact about set theory. Isabelle defines

$$\langle a, b\rangle \equiv \{\{a, a\}, \{a, b\}\},$$

which is equivalent but consists entirely of doubletons. This simplifies the proof — which we shall examine later — of the key property

$$\langle a,b\rangle = \langle c,d\rangle \leftrightarrow a=c \wedge b=d.$$

The next step is to define the projections, `fst` and `snd`. Descriptions are extremely useful here. We could put

$$\begin{aligned}\texttt{fst}(p) &\equiv \iota x\,.\,\exists y\,.\,p=\langle x,y\rangle\\ \texttt{snd}(p) &\equiv \iota y\,.\,\exists x\,.\,p=\langle x,y\rangle\end{aligned}$$

To show $\texttt{fst}(\langle a,b\rangle)=a$ by the rule (ι=), we must exhibit a unique x such that $\exists y\,.\,p=\langle x,y\rangle$ holds. Clearly $x=a$ (with $y=b$) by uniqueness of pairing. The treatment of `snd` is similar. Descriptions are suitable for defining many other kinds of destructors, such as case analysis operators for disjoint unions, natural numbers and lists. Isabelle's classical reasoner can prove the resulting equations.

Isabelle's ZF actually defines `fst` and `snd` indirectly. Following Martin-Löf's Constructive Type Theory [11], it defines the variable-binding projection $\texttt{split}(p,f)$, and proves the equation

$$\texttt{split}(\langle a,b\rangle,f)=f(a,b).$$

Frequently `split` is more convenient than the usual projections, which we can define concisely:[2]

$$\begin{aligned}\texttt{fst}(p) &\equiv \texttt{split}(p,\,x\,y\,.\,x)\\ \texttt{snd}(p) &\equiv \texttt{split}(p,\,x\,y\,.\,y)\end{aligned}$$

Like other destructors, `split` is defined using a description:

$$\texttt{split}(p,f) \equiv \iota z\,.\,\exists x\,y\,.\,p=\langle x,y\rangle \wedge z=f(x,y).$$

4.4 Cartesian products

The set $A\times B$ consists of all pairs $\langle a,b\rangle$ such that $a\in A$ and $b\in B$. Many authors [8, 22] define the Cartesian product in a cumbersome manner. If $a\in A$ and $b\in B$ then $\{\{a\},\{a,b\}\}\in\wp(\wp(A\cup B))$, so they define $A\times B$ using Separation:

$$A\times B \equiv \{z\in\wp(\wp(A\cup B))\,.\,\exists_{x\in A}\,.\,\exists_{y\in B}\,.\;z=\langle x,y\rangle\}$$

There is a historical and pedagogical case for this definition, which postpones the introduction of Replacement. But Replacement is built into our notation, so we might as well take advantage of it:

$$A\times B \equiv \bigcup_{x\in A}\bigcup_{y\in B}\{\langle x,y\rangle\}$$

[2]Here $x\,y\,.\,x$ and $x\,y\,.\,y$ stand for meta-level λ-abstractions, which would appear as `%x y.x` and `%x y.y` in an Isabelle source file.

This definition is self-evident, independent of the underlying representation of pairs, and easy to reason about.

Again, Isabelle actually defines $A \times B$ indirectly, following Martin-Löf's Type Theory. The disjoint union of a family of sets, $\sum_{x \in A} .B(x)$, is a useful generalization of $A \times B$. To generalize the definition above, we merely replace B by $B(x)$:

$$\sum_{x \in A} B(x) \equiv \bigcup_{x \in A} \bigcup_{y \in B(x)} \{\langle x, y \rangle\}$$

Natural deduction rules neatly summarize its properties:

$$\frac{a \in A \quad b \in B(a)}{\langle a, b \rangle \in (\sum_{x \in A} .B(x))} \ (\textstyle\sum I)$$

$$\frac{c \in (\sum_{x \in A} .B(x)) \qquad \begin{array}{c} [x \in A \quad y \in B(x) \quad c = \langle x, y \rangle]_{x,y} \\ \vdots \\ \theta \end{array}}{\theta} \ (\textstyle\sum E)$$

By $(\sum E)$, if $\langle a, b \rangle \in (\sum_{x \in A} .B(x))$ then $a \in A$ and $b \in B(a)$.

Now $A \times B$ is nothing but an abbreviation for $\sum_{x \in A} .B(x)$ when B involves no dependence upon x. Isabelle's parser and pretty printer handle these conventions.

4.5 Relations and functions

A **binary relation** is a set of ordered pairs. Isabelle's set theory defines the basic operations upon relations. These operations have the usual properties and require little discussion. Observe the usage of Replacement:

$$\begin{aligned}
\texttt{converse}(r) &\equiv \{z \,.\, w \in r,\, \exists x\, y \,.\, w = \langle x, y \rangle \wedge z = \langle y, x \rangle\} \\
\texttt{domain}(r) &\equiv \{x \,.\, w \in r,\, \exists y \,.\, w = \langle x, y \rangle\} \\
\texttt{range}(r) &\equiv \texttt{domain}(\texttt{converse}(r)) \\
\texttt{field}(r) &\equiv \texttt{domain}(r) \cup \texttt{range}(r)
\end{aligned}$$

Image and **inverse image** are infix operators:

$$\begin{aligned}
r \text{``} A &\equiv \{y \in \texttt{range}(r) \,.\, \exists_{x \in A} \,.\, \langle x, y \rangle \in r\} \\
r -\text{``} A &\equiv \texttt{converse}(r)\text{``}A
\end{aligned}$$

Functions are represented by their graphs, which are single-valued binary relations. The set of all functions from A to B is written $A \to B$. Just as we generalized $A \times B$ to $\sum_{x \in A} .B(x)$, we generalize $A \to B$ to $\prod_{x \in A} .B(x)$, the product of a family of sets. This concept predates Martin-Löf's Type Theory; it has a long history. We define

$$\prod_{x \in A} B(x) \equiv \{f \in \wp(\Sigma_{x \in A} \,.\, B(x)) \,.\, \forall_{x \in A} \,.\, \exists! y \,.\, \langle x, y \rangle \in f\}.$$

Here $A \to B$ abbreviates $\prod_{x\in A} .B(x)$ when B involves no dependence upon x. In particular, we have

$$(f \in A \to B) \leftrightarrow f \subseteq A \times B \land (\forall_{x\in A} . \exists! y . \langle x, y\rangle \in f).$$

We further define application and λ-abstraction. An explicit application operator is necessary; $f`a$ operates on the sets f and a. Observe how easily a description expresses the application operator:

$$\begin{aligned} f`a &\equiv \iota y . \langle a, y\rangle \in f \\ \lambda_{x\in A} . b(x) &\equiv \{\langle x, b(x)\rangle . x \in A\} \end{aligned}$$

Regarding functions as binary relations is tiresome. Only with difficulty can we derive high-level rules for functions, in the style of the λ-calculus.

$$\frac{\begin{matrix}[x \in A]_x \\ \vdots \\ b(x) \in B(x)\end{matrix}}{(\lambda_{x\in A} . b(x)) \in (\prod_{x\in A} .B(x))}\ (\lambda\Pi I) \qquad \frac{f \in (\prod_{x\in A} .B(x)) \quad a \in A}{f`a \in B(a)}\ (\lambda\Pi E)$$

$$\frac{a \in A}{(\lambda_{x\in A} . b(x))`a = b(a)}\ (\beta) \qquad \frac{f \in (\prod_{x\in A} .B(x))}{(\lambda_{x\in A} . f`x) = f}\ (\eta)$$

Injections, surjections and bijections are subsets of the total function space $A \to B$. Isabelle's set theory also defines composition of relations (including functions):

$$\begin{aligned} \mathtt{inj}(A, B) &\equiv \{f \in A \to B . \forall_{w\in A} . \forall_{x\in A} .\ f`w = f`x \to w = x\} \\ \mathtt{surj}(A, B) &\equiv \{f \in A \to B . \forall_{y\in B} . \exists_{x\in A} .\ f`x = y\} \\ \mathtt{bij}(A, B) &\equiv \mathtt{inj}(A, B) \cap \mathtt{surj}(A, B) \\ r \circ s &\equiv \{w \in \mathit{domain}(s) \times \mathit{range}(r) . \exists x\, y\, z . \\ &\qquad\qquad w = \langle x, z\rangle \land \langle x, y\rangle \in s \land \langle y, z\rangle \in r\} \end{aligned}$$

The numerous derived rules include

$$\frac{f \in \mathtt{bij}(A, B)}{\mathtt{converse}(f) \in \mathtt{bij}(B, A)} \qquad \frac{f \in \mathtt{inj}(A, B) \quad a \in A}{\mathtt{converse}(f)`(f`a) = a}$$

$$\frac{s \subseteq A \times B \quad r \subseteq B \times C}{(r \circ s) \subseteq A \times C} \qquad \frac{g \in A \to B \quad f \in B \to C}{(f \circ g) \in A \to C}$$

$$(r \circ s) \circ t = r \circ (s \circ t)$$

Thus, relations and functions are closed under composition. A similar property is proved for injections, surjections and bijections.

5 Examples of set-theoretic reasoning

To give some idea of the level of reasoning possible in Isabelle, we shall examine three simple examples: ordered pairing, Cantor's Theorem, and the Composition of Homomorphisms challenge [3]. The sessions given below are based on polished proofs from Isabelle's set theory. I have simplified the commands to make the proofs slightly longer and easier to follow.

This section, which is intended for casual reading, describes the effect of each command in general terms. For details of the many Isabelle primitives that appear, please consult the documentation [15].

5.1 Injectivity of ordered pairing

Proving that $\langle a,b\rangle \equiv \{\{a,a\},\{a,b\}\}$ is a valid definition of ordered pairing is tiresome — see Halmos [8, page 23], for example. Here is a short machine proof using Isabelle's tools. We do not see all the details of a full proof (that happens internally) but we do see the key lemma. We now state this lemma, which concerns doubletons, to Isabelle:

```
goal ZF_Rule.thy "{a,b} = {c,d}  <->  (a=c & b=d) | (a=d & b=c)";
  Level 0
  {a,b} = {c,d} <-> a = c & b = d | a = d & b = c
   1. {a,b} = {c,d} <-> a = c & b = d | a = d & b = c
```

This is the initial state of a backward proof. It has one subgoal, which is the same as the main or ultimate goal. Our first inference will apply the derived rule

$$\frac{P \leftrightarrow Q \quad Q \leftrightarrow R}{P \leftrightarrow R}$$

to let us replace $\{a,b\} = \{c,d\}$ by any equivalent formula:

```
by (resolve_tac [iff_trans] 1);
  Level 1
  {a,b} = {c,d} <-> a = c & b = d | a = d & b = c
   1. {a,b} = {c,d} <-> ?Q
   2. ?Q <-> a = c & b = d | a = d & b = c
```

The one subgoal has become two, and the unknown intermediate formula appears as `?Q`. The first occurrence of $=$ in the main goal is one of the rare cases when the Axiom of Extensionality is directly useful. We replace $\{a,b\} = \{c,d\}$ by the inclusions $\{a,b\} \subseteq \{c,d\}$ and $\{c,d\} \subseteq \{a,b\}$, updating `?Q`.

```
by (resolve_tac [extension] 1);
  Level 2
  {a,b} = {c,d} <-> a = c & b = d | a = d & b = c
   1. {a,b} <= {c,d} & {c,d} <= {a,b} <-> a = c & b = d | a = d & b = c
```

Subgoal 1 has vanished; subgoal 2 has taken its place; `?Q` has become the conjunction of inclusions. The remaining subgoal requires a massive but essentially trivial case analysis. If $\{a,b\} \subseteq \{c,d\}$ then the rule $(\subseteq E)$ states that if $x \in \{a,b\}$ then $x \in \{c,d\}$; putting $x = a$ we obtain $a = c \lor a = d$, and so forth. (Halmos's proof makes a much smaller case analysis.) The classical tactic `fast_tac` proves the subgoal. It takes the collection of natural deduction rules proved so far, packaged as `upair_cs`.

```
by (fast_tac upair_cs 1);
  Level 3
  {a,b} = {c,d} <-> a = c & b = d | a = d & b = c
  No subgoals!
```

This automatic step takes about nine seconds.[3] Finally, we declare the resulting theorem as the ML identifier `doubleton_iff`:

```
val doubleton_iff = result();
```

Now we prove the main theorem, that ordered pairing is injective. While stating the goal, we make Isabelle expand the definition `Pair_def`:

```
goalw ZF_Rule.thy [Pair_def] "<a,b> = <c,d>  <->  a=c & b=d";
  Level 0
  <a,b> = <c,d> <-> a = c & b = d
   1. {{a,a},{a,b}} = {{c,c},{c,d}} <-> a = c & b = d
```

The expanded subgoal 1 is full of doubletons. We rewrite it using our lemma (`FOL_ss` is a collection of standard rewrite rules for first-order logic):

```
by (SIMP_TAC (FOL_ss addrews [doubleton_iff]) 1);
  Level 1
  <a,b> = <c,d> <-> a = c & b = d
   1. a = c & (b = d | c = d & b = d) |
      (a = c & c = d | a = d & d = c) & a = c & b = c <->
      a = c & b = d
```

The easiest way to prove the resulting subgoal involves further case analysis. This time, `fast_tac` requires only the rules of first-order logic, although supplying additional rules would do no harm.

```
by (fast_tac FOL_cs 1);
  Level 2
  <a,b> = <c,d> <-> a = c & b = d
  No subgoals!
```

Given the lemma, the total time to prove this theorem is about three seconds.

[3]All Isabelle timings are on a Sun SPARCstation ELC.

5.2 Cantor's Theorem

Cantor's Theorem is one of the few major results in mathematics that can be proved automatically [1]. It is easily expressed and its proof, although deep, is short.

```
goal ZF_Rule.thy "ALL f: A->Pow(A). EX S: Pow(A). ALL x:A. ~ f`x=S";
  Level 0
  ALL f:A -> Pow(A). EX S:Pow(A). ALL x:A. ~ f ` x = S
   1. ALL f:A -> Pow(A). EX S:Pow(A). ALL x:A. ~ f ` x = S
```

We begin by routine rule applications, using the introduction rules for the bounded quantifiers:

```
by (resolve_tac [ballI] 1);
  Level 1
  ALL f:A -> Pow(A). EX S:Pow(A). ALL x:A. ~ f ` x = S
   1. !!f. f : A -> Pow(A) ==> EX S:Pow(A). ALL x:A. ~ f ` x = S
```

Subgoal 1 requires showing $\exists_{S\in\wp(A)} . \forall_{x\in A} . f`x \neq S$ under the assumption $f \in A \rightarrow \wp(A)$, where f is arbitrary.

```
by (resolve_tac [bexI] 1);
  Level 2
  ALL f:A -> Pow(A). EX S:Pow(A). ALL x:A. ~ f ` x = S
   1. !!f. f : A -> Pow(A) ==> ALL x:A. ~ f ` x = ?S1(f)
   2. !!f. f : A -> Pow(A) ==> ?S1(f) : Pow(A)
```

Under the same assumption, we now have two subgoals. The first, crucial goal involves the term `?S1(f)`, which is a placeholder for something that may depend upon f. Proving the subgoal instantiates this term with Cantor's diagonal set.

We can prove it automatically with `best_tac`, a classical reasoning tactic that employs best-first search. The search space is large and undirected. We must supply `best_tac` with a minimal collection of rules — though some readers might regard this as cheating.

```
val cantor_cs = FOL_cs
  addSIs [ballI, CollectI, PowI, subsetI] addIs [bexI]
  addSEs [CollectE, equalityCE];
```

Starting with `FOL_cs` — the rules for first-order logic — we add rules for the bounded quantifiers, powersets, the subset relation, Separation and extensional equality.

```
 by (best_tac cantor_cs 1);
  Level 3
  ALL f:A -> Pow(A). EX S:Pow(A). ALL x:A. ~ f ` x = S
   1. !!f. f : A -> Pow(A) ==> {x: A . ~ x : f ` x} : Pow(A)
```

After six seconds, we have obtained the diagonal set, which is $\{x \in A . x \notin f`x\}$. The remaining subgoal is to show that the diagonal set belongs to $\wp(A)$. This is trivial; we may employ depth-first search (via `fast_tac`) and supply a large collection of rules (ZF_cs):

```
by (fast_tac ZF_cs 1);
  Level 4
  ALL f:A -> Pow(A). EX S:Pow(A). ALL x:A. ~ f ‘ x = S
  No subgoals!
```

Quaife [18, page 114] remarks that Otter could not construct the diagonal set; we have just seen Isabelle do so. Indeed, we could have proved Cantor's Theorem by a single call to best_tac. However, the classical reasoner is not designed to cope with such undirected searches. Equivalent forms of Cantor's Theorem cause the search to founder, even using the minimal collection of rules cantor_cs.

5.3 Composition of homomorphisms

Boyer et al. [3] posed this as a challenge problem, and supplied a hand proof involving twenty-seven lemmas. Proving the theorem from the axioms alone might indeed be a challenge, but I found it easy in Isabelle's set theory. The proof effort took about half an hour, much of which was spent keying in and correcting the conjecture. Most of the twenty-seven lemmas were already proved in Isabelle's set theory. Five of them concerned proving that some class is a set, which is never necessary in ZF. Others were perhaps proved on-the-fly by Isabelle's simplifier. My proof required no explicit lemmas.

Their definition of homomorphism can be put into a more conventional notation (making the problem slightly harder!) by making $\mathrm{hom}(A, f, B, g)$ denote the set of all homomorphisms from A to B:

$$\begin{aligned}\mathrm{hom}(A, f, B, g) \quad \equiv \quad & \{H \in A \to B\,.\,(f \in A \times A \to A) \wedge (g \in B \times B \to B) \wedge \\ & \qquad (\forall_{x\in A}\,.\,\forall_{y\in A}\,.\,H`(f`\langle x, y\rangle) = g`\langle H`x, H`y\rangle)\}\end{aligned}$$

The contrast between the previous example and this one is clear. Cantor's Theorem is fundamental; its proof is short, but difficult to find. The fact that homomorphisms are closed under composition is straightforward, but has a long proof. The proof is mainly by rewriting, with some propositional reasoning to break up the conjunctions. We can set up SIMP_TAC such that it calls fast_tac to prove its rewritten formulae, even when trying conditional rewrite rules; a single invocation of SIMP_TAC proves the theorem in about thirty-three seconds. But the proof is easier to follow if we perform it several steps.

First we state the goal, binding the definition of homomorphism to the ML identifier `hom_def`:

```
val [hom_def] = goal Perm.thy
   "(!! A f B g. hom(A,f,B,g) ==                                     \
\       {H: A->B. f:A*A->A & g:B*B->B &                              \
\            (ALL x:A. ALL y:A. H`(f`<x,y>) = g`<H`x,H`y>)}) ==> \
\   J : hom(A,f,B,g) & K : hom(B,g,C,h) -->                          \
\   (K O J) : hom(A,f,C,h)";
  Level 0
  J : hom(A,f,B,g) & K : hom(B,g,C,h) --> K O J : hom(A,f,C,h)
   1. J : hom(A,f,B,g) & K : hom(B,g,C,h) --> K O J : hom(A,f,C,h)
```

Next, we expand `hom_def` in the subgoal:

```
by (rewtac hom_def);
  Level 1
  J : hom(A,f,B,g) & K : hom(B,g,C,h) --> K O J : hom(A,f,C,h)
   1. J :
      {H: A -> B .
       f : A * A -> A &
       g : B * B -> B &
       (ALL x:A. ALL y:A. H ` (f ` <x,y>) = g ` <H ` x,H ` y>)} &
      K :
      {H: B -> C .
       g : B * B -> B &
       h : C * C -> C &
       (ALL x:B. ALL y:B. H ` (g ` <x,y>) = h ` <H ` x,H ` y>)} -->
      K O J :
      {H: A -> C .
       f : A * A -> A &
       h : C * C -> C &
       (ALL x:A. ALL y:A. H ` (f ` <x,y>) = h ` <H ` x,H ` y>)}
```

Next we invoke a simple tactic from the classical reasoner, in order to break up conjunctions and remove the instances of Separation:

```
by (safe_tac ZF_cs);
  Level 2
  J : hom(A,f,B,g) & K : hom(B,g,C,h) --> K O J : hom(A,f,C,h)
   1. [| J : A -> B; K : B -> C; f : A * A -> A; g : B * B -> B;
         g : B * B -> B;
         ALL x:A. ALL y:A. J ` (f ` <x,y>) = g ` <J ` x,J ` y>;
         h : C * C -> C;
         ALL x:B. ALL y:B. K ` (g ` <x,y>) = h ` <K ` x,K ` y> |] ==>
      K O J : A -> C
   2. !!x y.
         [| J : A -> B; K : B -> C; f : A * A -> A; g : B * B -> B;
            g : B * B -> B;
            ALL x:A. ALL y:A. J ` (f ` <x,y>) = g ` <J ` x,J ` y>;
            h : C * C -> C;
            ALL x:B. ALL y:B. K ` (g ` <x,y>) = h ` <K ` x,K ` y>; x : A;
            y : A |] ==>
         (K O J) ` (f ` <x,y>) = h ` <(K O J) ` x,(K O J) ` y>
```

Next, we collect some rewrites to supply to the simplifier. The collection need not be minimal, so we begin with `ZF_ss` (a standard collection of rewrite rules) and add four relevant lemmas:

```
val hom_ss =
  ZF_ss addrews [comp_func,comp_func_apply,SigmaI,apply_type]
        addcongs (mk_congs Perm.thy ["op O"]);
```

Subgoal 1 is one of the lemmas, namely that functions are closed under composition. Because simplification must employ the assumptions, in particular $J \in A \to B$ and $K \in B \to C$, the correct tactic here is `ASM_SIMP_TAC`:

```
by (ASM_SIMP_TAC hom_ss 1);
  Level 3
  J : hom(A,f,B,g) & K : hom(B,g,C,h) --> K O J : hom(A,f,C,h)
   1. !!x y.
         [| J : A -> B; K : B -> C; f : A * A -> A; g : B * B -> B;
            g : B * B -> B;
            ALL x:A. ALL y:A. J ` (f ` <x,y>) = g ` <J ` x,J ` y>;
            h : C * C -> C;
            ALL x:B. ALL y:B. K ` (g ` <x,y>) = h ` <K ` x,K ` y>; x : A;
            y : A |] ==>
         (K O J) ` (f ` <x,y>) = h ` <(K O J) ` x,(K O J) ` y>
```

Finally, we must show that $K \circ J$ maps applications of f to applications of h. The simplifier applies the rewrite

$$\frac{g \in A \to B \quad f \in B \to C \quad a \in A}{(f \circ g)`a = f`(g`a)}$$

and uses the quantified assumptions about J and K as further rewrites. These rewrites are all conditional. The simplifier verifies the conditions using lemmas and the assumptions; this is essentially type checking.

```
by (ASM_SIMP_TAC hom_ss 1);
  Level 4
  J : hom(A,f,B,g) & K : hom(B,g,C,h) --> K O J : hom(A,f,C,h)
  No subgoals!
```

The total time for this simple proof is under sixteen seconds.

6 Ramsey's Theorem in ZF

Ramsey's Theorem is a profound generalization of the pigeon-hole principle. A special case of it, the finite exponent 2 version, has become something of a benchmark for theorem provers. Basin and Kaufman [2] compare proofs of this result using the Boyer/Moore Theorem Prover (called NQTHM) and Nuprl. The theorem is an informative example because its proof is both deep and long, involving graphs, sets and natural numbers. It covers a broad spectrum of reasoning issues. It is no toy example, but a major theorem with serious applications.

NQTHM and Nuprl are utterly different; they hardly admit a meaningful comparison. Isabelle with ZF set theory is much closer to Nuprl: both support interactive, goal-directed proof using tactics and tacticals; both employ full predicate logic and some form of set theory. But Nuprl implements Martin-Löf's Constructive Type Theory rather than classical set theory.

The NQTHM and Nuprl proofs both represent finite sets by lists without repetitions. This representation has many disadvantages: it does not handle infinite sets; union and intersection do not satisfy many of the usual equations; in a constructive or computational setting, it requires an equality test for the elements. The Isabelle proof represents sets by sets.

6.1 The natural numbers in Isabelle's set theory

In set theory, the natural number n is the n-element set $\{0, \ldots, n-1\}$. The companion paper will describe the construction of the set of natural numbers, and the derivation of recursion and induction. Isabelle's set theory proves many facts in elementary arithmetic. Here is a summary of the things needed for Ramsey's Theorem. The addition and subtraction operators are $\oplus$ and $\ominus$ because $+$ and $-$ stand for disjoint union and set difference, respectively.

`nat`	set of natural numbers
0	zero (identical to $\emptyset$, the empty set)
$m \oplus n$	sum of the natural numbers m and n
$m \ominus n$	difference of the natural numbers m and n
`succ`(m)	the successor of m, namely $m \oplus 1$

6.2 The definitions in ZF

Rather than attempt to improve Basin and Kaufmann's description of Ramsey's Theorem, I briefly discuss the corresponding definitions. I have used these largely as abbreviations, rather than as abstract notions; in most of the Isabelle proofs, the definitions are expanded.

Basin and Kaufmann's version of the theorem requires the notion of undirected graph, whose edge set E is a symmetric binary relation. Sets of *unordered* pairs, instead of symmetric relations, would be more in harmony with the general finite version of Ramsey's Theorem [19].

$$\texttt{Symmetric}(E) \equiv \forall x\, y\,.\, \langle x,y\rangle \in E \to \langle y,x\rangle \in E$$

Let V be a set of vertices and E a symmetric edge relation. Then C is a **clique** if $C \subseteq V$ and every pair of distinct nodes in C is joined by an edge in E. Dually, I is an **independent set** (or **anticlique**) if $I \subseteq V$ and no pair of distinct nodes in I is joined by an edge in E.

$$\begin{aligned}\texttt{Clique}(C,V,E) &\equiv C \subseteq V \wedge (\forall_{x\in C}\,.\,\forall_{y\in C}\,.\, x \neq y \to \langle x,y\rangle \in E)\\ \texttt{Indept}(I,V,E) &\equiv I \subseteq V \wedge (\forall_{x\in I}\,.\,\forall_{y\in I}\,.\, x \neq y \to \langle x,y\rangle \notin E)\end{aligned}$$

Most of my efforts went to proving results that properly belong to a theory of cardinality. Representing sets by lists without repetitions would have an advantage here: their cardinality is simply their length and many facts can be proved by routine inductions. At present, Isabelle's set theory does not define cardinality. Fortunately, the Ramsey proof requires only the notion 'S has at least n elements.' This is equivalent to 'there is an injection from n to S' because the natural number n has n elements:

$$\texttt{Atleast}(n,S) \equiv \exists f\,.\, f \in \mathit{inj}(n,S)$$

Finally, we define an abbreviation for Ramsey's Theorem:

$$\begin{aligned}\texttt{Ramsey}(n,i,j) \equiv \forall V\, E\,.\, &\texttt{Symmetric}(E) \wedge \texttt{Atleast}(n,V) \to\\ &(\exists C\,.\,\texttt{Clique}(C,V,E) \wedge \texttt{Atleast}(i,C)) \vee\\ &(\exists I\,.\,\texttt{Indept}(I,V,E) \wedge \texttt{Atleast}(j,I))\end{aligned}$$

Now Ramsey's Theorem is easily stated:

$$\frac{i \in \texttt{nat} \quad j \in \texttt{nat}}{\exists_{n\in\texttt{nat}}\, \texttt{Ramsey}(n,i,j)}$$

Originally I defined

$$\texttt{Graph}(V,E) \equiv (E \subseteq V \times V) \wedge \texttt{Symmetric}(E)$$

and put $\texttt{Graph}(V,E)$ instead of $\texttt{Symmetric}(E)$ in the definition of $\texttt{Ramsey}$, but this was a needless complication. Since E is universally quantified, the assertion holds for all E, including those such that $E \subseteq V \times V$.

All the lemmas proved for Ramsey's Theorem — except five that have been moved to the general library — are discussed below. Many are proved automatically.

6.3 Cliques and independent sets

The classical reasoner (`fast_tac`) proves these four facts automatically, taking just over one second in total.

$$\texttt{Clique}(\emptyset, V, E) \qquad \texttt{Indept}(\emptyset, V, E)$$

$$\frac{\texttt{Clique}(C, V', E) \quad V' \subseteq V}{\texttt{Clique}(C, V, E)} \qquad \frac{\texttt{Indept}(I, V', E) \quad V' \subseteq V}{\texttt{Indept}(I, V, E)}$$

6.4 Cardinality

The classical reasoner automatically proves (in under two seconds) that every set has at least zero elements:

$$\texttt{Atleast}(0, A)$$

A useful rule for induction steps is derived by six explicit rule applications:

$$\frac{\texttt{Atleast}(\texttt{succ}(m), A)}{\exists_{x \in A} \,.\, \texttt{Atleast}(m, A - \{x\})}$$

A property of subsets has a short proof, using a related fact about injections:

$$\frac{\texttt{Atleast}(n, A) \quad A \subseteq B}{\texttt{Atleast}(n, B)}$$

This rule for adding an element to a set (`cons`) is proved by five rule applications:

$$\frac{\texttt{Atleast}(m, B) \quad b \notin B}{\texttt{Atleast}(\texttt{succ}(m), \texttt{cons}(b, B))}$$

Using `fast_tac` and the previous two results quickly yields

$$\frac{\texttt{Atleast}(m, B - \{x\}) \quad x \in B}{\texttt{Atleast}(\texttt{succ}(m), B)}$$

The following theorem is the pigeon-hole principle for two pigeon-holes. Proving it took up most of the time I devoted to Ramsey's Theorem. The proof involves induction on m and n with several case analyses; it consists of a complex mixture of proof checking with the tools `fast_tac` and `ASM_SIMP_TAC`.

$$\frac{m \in \texttt{nat} \quad n \in \texttt{nat} \quad \texttt{Atleast}(m \oplus n, A \cup B)}{\texttt{Atleast}(m, A) \vee \texttt{Atleast}(n, B)}$$

6.5 Ramsey's Theorem: the inductive argument

Ramsey's Theorem requires a double induction. Using previous lemmas, `fast_tac` proves the two base cases automatically (taking under three seconds in total):

$$\texttt{Ramsey}(0,0,j) \qquad \texttt{Ramsey}(0,i,0)$$

Before we can tackle the induction step, we must prove three lemmas. The first is an instance of the pigeon-hole principle:

$$\frac{\texttt{Atleast}(m \oplus n, A) \quad m \in \texttt{nat} \quad n \in \texttt{nat}}{\texttt{Atleast}(m, \{x \in A \,.\, \neg P(x)\}) \vee \texttt{Atleast}(n, \{x \in A \,.\, P(x)\})}$$

The next two lemmas contain the key idea of Ramsey's Theorem. One gives a method of extending a certain independent set of size j to one of size $\texttt{succ}(j)$; the other gives a similar method for cliques. Using the definitions of `Symmetric`, `Indept`, and `Clique`, the standard rules (`ZF_cs`), and a lemma above concerning `Atleast`, `fast_tac` proves both theorems automatically! Each proof takes roughly one minute, accounting for two-thirds of the CPU time in the entire proof.

$$\frac{\texttt{Symmetric}(E) \quad \texttt{Indept}(I, \{z \in V - \{a\} \,.\, \langle a,z\rangle \notin E\}, E) \quad a \in V \quad \texttt{Atleast}(j, I)}{\texttt{Indept}(\texttt{cons}(a,I), V, E) \wedge \texttt{Atleast}(\texttt{succ}(j), \texttt{cons}(a,I))}$$

$$\frac{\texttt{Symmetric}(E) \quad \texttt{Clique}(C, \{z \in V - \{a\} \,.\, \langle a,z\rangle \in E\}, E) \quad a \in V \quad \texttt{Atleast}(j, C)}{\texttt{Clique}(\texttt{cons}(a,C), V, E) \wedge \texttt{Atleast}(\texttt{succ}(j), \texttt{cons}(a,C))}$$

The induction step of Ramsey's Theorem is tedious, even with all the lemmas. The proof involves a four-way case split, with many explicit rule applications as well as invocations of the classical reasoner:

$$\frac{\texttt{Ramsey}(m, \texttt{succ}(i), j) \quad \texttt{Ramsey}(n, i, \texttt{succ}(j)) \quad m \in \texttt{nat} \quad n \in \texttt{nat}}{\texttt{Ramsey}(\texttt{succ}(m \oplus n), \texttt{succ}(i), \texttt{succ}(j))}$$

Finally, we prove the Theorem itself. This involves performing the double induction, invoking lemmas for the base cases and induction step:

$$\frac{i \in \texttt{nat} \quad j \in \texttt{nat}}{\exists_{n \in \texttt{nat}} \texttt{Ramsey}(n, i, j)}$$

6.6 Discussion and comparison

The induction step and base cases constitute a PROLOG program for $\texttt{Ramsey}(n,i,j)$, which we may express in a functional style:

$$\begin{aligned} r(0,j) &= 0 \\ r(i,0) &= 0 \\ r(i+1,j+1) &= r(i+1,j) + r(i,j+1) + 1 \end{aligned}$$

Since $r(i,j)$ computes a number n satisfying $\texttt{Ramsey}(n,i,j)$, it is called the **witnessing function** for Ramsey's Theorem. Basin and Kaufman [2] obtain slightly different Ramsey numbers; the definitions reflect details of the proofs.

Nuprl expresses Ramsey's Theorem using quantifiers, as here. Since its logic is constructive, Nuprl can extract a witnessing function from the proof. NQTHM lacks quantifiers; it expresses Ramsey's Theorem in terms of a witnessing function, obtained from a hand proof. Both the Nuprl and NQTHM proofs involve additional witnessing functions, which map a graph of sufficient size to a clique or independent set. The Isabelle proof follows the same reasoning as Basin and Kaufman's proofs; it does not make essential use of classical logic. Because it is conducted in classical ZF set theory, there is no way of extracting such witnessing functions from the proof.

The table compares the NQTHM, Nuprl and Isabelle/ZF proofs:

	NQTHM	Nuprl	Isabelle
# Tokens	933	972	975
# Definitions	10	24	5
# Lemmas	26	25	17
# Replay Time	3.7 minutes (Sun 3/60)	57 minutes (Symbolics 3670)	3.2 minutes (SPARC ELC)

The figures for Isabelle include all the definitions and lemmas given above, and their proofs. The Isabelle proof has the fewest definitions and lemmas. But NQTHM has by far the shortest replay time, since a Sun SPARCstation ELC is three or four times faster than a Sun 3/60. Kaufman took seven hours to find the NQTHM proof; Basin required twenty hours, plus a further sixty for library development [2]. I took about nine hours to develop the Isabelle proof, including all lemmas.

Tokens were counted, after removal of comments, by the Unix command

```
sed -e "s/[^A-Za-z0-9'_]/ /g" ramsey.ML | wc
```

This counts identifiers but not symbols such as `:` and `=`, and is therefore an underestimate. It counts seven tokens in `EX x:A. Atleast(m, A-{x})`. Basin and Kaufman each used different methods for counting tokens in their proofs. Figure 1 gives a more pessimistic impression of the token density of Isabelle proofs. One theorem is proved automatically. Another, which is the main induction step, has the second longest proof of the entire effort. The third is Ramsey's Theorem itself, with its inductions on i and j.

Comparisons are difficult. There are discrepancies in the hardware, token counting methods, etc. Furthermore, each author of a proof was an expert with his system. We can hardly predict how the systems would compare if tested by novices. The proof requires familiarity with both the system and its library of theorems.

Given these reservations, what conclusions can we draw? Isabelle stands up well against two extensively developed systems, despite its lack of arithmetic decision procedures and small size (about 9000 lines of Standard ML, excluding object-logic definitions). More importantly, the ZF proof demolishes the myth that axiomatic set theory is too cumbersome to use. Its formal language can be made clear and

```
val prems = goalw Ramsey.thy [Symmetric_def,Clique_def]
    "[| Symmetric(E);  Clique(C, {z: V-{a}. <a,z>:E}, E);  a: V;    \
\       Atleast(j,C) |] ==>                                         \
\    Clique(cons(a,C), V, E) & Atleast(succ(j), cons(a,C))";
by (cut_facts_tac prems 1);
by (fast_tac (ZF_cs addSEs [Atleast_succI]) 1);
val Clique_succ = result();


val ram1::ram2::prems = goalw Ramsey.thy [Ramsey_def]
   "[| Ramsey(m,succ(i),j);  Ramsey(n,i,succ(j));  m:nat;  n:nat |] ==> \
\   Ramsey(succ(m#+n), succ(i), succ(j))";
by (safe_tac ZF_cs);
by (etac (Atleast_succD RS bexE) 1);
by (eres_inst_tac [("P1","%z.<x,z>:E")] (Atleast_partition RS disjE) 1);
by (REPEAT (resolve_tac prems 1));
(*case m*)
by (rtac (ram1 RS spec RS spec RS mp RS disjE) 1);
by (fast_tac ZF_cs 1);
by (fast_tac (ZF_cs addEs [Clique_superset]) 1); (*we have a Clique*)
by (safe_tac ZF_cs);
by (eresolve_tac (swapify [exI]) 1);
by (REPEAT (ares_tac [Indept_succ] 1));          (*make a bigger Indept*)
(*case n*)
by (rtac (ram2 RS spec RS spec RS mp RS disjE) 1);
by (fast_tac ZF_cs 1);
by (safe_tac ZF_cs);
by (rtac exI 1);
by (REPEAT (ares_tac [Clique_succ] 1));          (*make a bigger Clique*)
by (fast_tac (ZF_cs addEs [Indept_superset]) 1); (*we have an Indept*)
val Ramsey_step_lemma = result();


val prems = goal Ramsey.thy
    "i: nat ==> ALL j: nat. EX n:nat. Ramsey(n,i,j)";
by (nat_ind_tac "i" prems 1);
by (fast_tac (ZF_cs addSIs [nat_0_I,Ramsey00j]) 1);
by (rtac ballI 1);
by (nat_ind_tac "j" [] 1);
by (fast_tac (ZF_cs addSIs [nat_0_I,Ramsey0i0]) 1);
by (dres_inst_tac [("x","succ(j1)")] bspec 1);
by (REPEAT (eresolve_tac [nat_succ_I,bexE] 1));
by (rtac bexI 1);
by (rtac Ramsey_step_lemma 1);
by (REPEAT (ares_tac [nat_succ_I,add_type] 1));
val ramsey = result();
```

Figure 1: Part of the Isabelle proof of Ramsey's Theorem

natural, and Isabelle's tools — though far from perfect — allow proofs to proceed in large steps.

7 Previous work using Isabelle

Isabelle has supported some form of ZF set theory since its early days. My original version consisted of idiosyncratic axioms over the classical sequent calculus LK, with derived sequent rules for the set constructors [13, page 382].

When Philippe Nöel started working in set theory, he found both the axioms and the sequent calculus uncongenial. He adopted Suppes's axioms and natural deduction (then newly available). Isabelle's set theory was developed only up to ordered pairs. Nöel went on to prove a large body of results: theorems about relations, functions, orderings, fixed points, recursion, and more [10].

His priority was to develop as much mathematics as possible, not to create short and elegant proofs. Many of his proofs comprised ten, fifty or even 100 tactic steps. Tactical proofs are a form of software; the simpler they are, the easier they are to understand and maintain. Martin Coen adopted some of Nöel's proofs for his own use [5]. He polished them a bit, but much more remained to be done.

The present work is an attempt to make set theory easy. Simple facts should have simple proofs: a few rule applications or tool invocations. This has largely been accomplished; proofs are easily an order of magnitude simpler than before. Here are some techniques for taming set theory.

7.1 Definitions and natural deduction

Many authors expand definitions heavily, though the resulting formula is likely to be unreadable, if not enormous. Recall that binary intersection is defined by

$$A \cap B \equiv \bigcap(\mathtt{Upair}(A, B)).$$

This may seem simple enough, but $\bigcap$ is defined in terms of $\bigcup$ and Separation; Separation and `Upair` are both defined in terms of Replacement, and so forth. Expanding definitions reduces a set-theoretic assertion to one in first-order logic, but at the cost of destroying all intuition.

The alternative to expanding definitions is deriving additional lemmas or rules. Natural deduction is a style in which each rule describes how to introduce or eliminate some constant. Ideally, each rule should mention only one constant. Repeatedly applying such rules analyses a formula, breaking it down to atomic subformulae; this can be automated.

The natural deduction style constrains the form of each rule, and provides a naming convention. Thus, it is a powerful tool for organizing what might otherwise become a haphazard collection of lemmas. We are practically forced to derive the following natural deduction rules for intersection:

$$\frac{c \in A \quad c \in B}{c \in A \cap B}\ (\cap I) \qquad \frac{c \in A \cap B}{c \in A}\ (\cap E1) \qquad \frac{c \in A \cap B}{c \in B}\ (\cap E2)$$

Intersection could instead be defined by

$$A \cap B \equiv \{x \in A . x \in B\}.$$

Adopting this definition would affect the derivations of the rules $(\cap I)$, $(\cap E1)$, and $(\cap E2)$, but other proofs would be unaffected. For instance, $A \cap B = B \cap A$ has a simple proof using the natural deduction rules. Schmidt [21] also argues the case for natural deduction in set theory.

7.2 Descriptions

The description $\iota x . \psi(x)$ is perfectly innocuous, being nothing but a name for the unique object a satisfying $\psi(a)$, if such exists. Descriptions are seldom mentioned in the literature, yet they are much more convenient than direct calculatons. We can define the first projection by

$$\mathtt{fst}(p) \equiv \iota x . \exists y . p = \langle x, y \rangle$$

instead of Noël's

$$\mathtt{fst}(p) \equiv \bigcup(\bigcap(p)).$$

The former definition is independent of the representation of ordered pairing; to show $\mathtt{fst}(\langle a, b\rangle) = a$, we simply appeal to a previous theorem about the injectivity of $\langle a, b\rangle$. The latter definition requires proving $\bigcup(\bigcap(\{\{a, a\}, \{a, b\}\})) = a$. The second projection (`snd`) can be defined easily using ι, but otherwise requires a complex expression.

As a general remark, improving the primitive operations pays handsomely. Replacement became easy to use after I derived a version with a simpler single-valued condition (§3.4). The new form of Replacement afforded improvements to existing definitions. For instance, Nöel defined the domain of a relation using Separation:

$$\mathtt{domain}(r) \equiv \{x \in \bigcup(\bigcup(r)) . \exists y . \langle x, y\rangle \in r\}$$

When Replacement became easy to use, I adopted

$$\mathtt{domain}(r) \equiv \{x . w \in r, \exists y . w = \langle x, y\rangle\}.$$

This is more concise, and is independent of the representation of ordered pair.[4]

7.3 Tool development

When Nöel started his work, Isabelle provided little automation. There was a crude simplifier and the classical reasoner was difficult to invoke. Nöel developed a tactic that could prove many of his simpler theorems automatically [10]. It worked by expanding definitions.

[4]Unless r is known to be a binary relation, $\{\mathtt{fst}(w) . w \in r\}$ is not equivalent to $\mathtt{domain}(r)$.

Much later, I modified Isabelle's classical reasoner to be generic, and suitable for set theory. To help prevent subgoals of the form $t \in ?A$ from causing runaway instantiations (see §3.1), I reordered the premises of some rules. (Premises create subgoals, which are normally processed from left to right, like in Prolog.) Finally, I extended Isabelle with ways of preventing the instantiation of unknowns in subgoals.

Also during this period, Tobias Nipkow installed his simplifier [9].

Specialized rewriters and theorem provers may be much faster, but Isabelle's tools offer satisfactory performance: they normally return in a few seconds. Because my proof style minimizes the expanding of definitions, defining new concepts does not make proofs slower.

Tools obviously improve user productivity; moreover, the resulting proofs are resilient. Proof checking causes brittleness: proofs 'break' (fail to replay) after the slightest change to a definition or axiom. Tools generally adapt to changes. For a striking example of resilience, recall the pigeon-hole principle:

$$\frac{m \in \mathtt{nat} \quad n \in \mathtt{nat} \quad \mathtt{Atleast}(m \oplus n, A \cup B)}{\mathtt{Atleast}(m, A) \vee \mathtt{Atleast}(n, B)}$$

The lemma can be strengthened: replace $m \oplus n$ by $m \oplus n \ominus 1$, where $1 \equiv \mathtt{succ}(0)$. When I did this, the previous proof (consisting of twenty-eight commands!) replayed perfectly. The nested inductions went precisely as before; the case analyses were identical. The $\cdots \ominus \mathtt{succ}(0)$ caused no difficulties because all subgoals containing it were submitted to the simplifier, using a general collection of arithmetic rewrites. This was partly luck, but the new version of the pigeon-hole principle required only slight changes to the rest of Ramsey's Theorem.

8 Conclusions

Isabelle's version of ZF set theory, with its definitions, derived rules and tools, has reached an advanced sate of development. Problems can be stated in a reasonably familiar notation and approached using high-level steps.

Quaife [18] and Saaltink [20] have also performed extensive proofs in axiomatic set theory. Quaife uses NBG set theory. He has obtained a degree of automation from the resolution theorem prover Otter; this requires proving a suitable series of lemmas, sometimes stated in a technical form, and carefully assigning weights and other settings of Otter. Saaltink uses the Eves theorem prover, which has the ZF axioms built in. His proofs consist of commands to the Eves reducer, which can expand definitions and perform various simplifications.

I would not attempt an objective comparison with Quaife's work — the culture gap between Isabelle and Otter is too great. But ZF is much more to my taste than NBG. Quaife forgoes the notations $\{x \in A \,.\, \psi(x)\}$ and $\iota x \,.\, \psi(x)$, which seem essential for clarity. Saaltink's work uses ZF and its interactive proof style resembles Isabelle's.

Boyer et al. [3] and Quaife mention the possibility of theorem provers settling famous open questions such as Goldbach's Conjecture. This seems unlikely in the near future, especially since some of these open questions may be independent of the axioms of set theory. A more immediate goal is to produce a reasoning tool to assist mathematicians, just as symbolic algebra packages assist engineers. Even this modest goal requires more research. Set theory by itself does not support mathematical abstraction — the set-theoretic definition of group leads to a horrendous syntax for group theory. This is an area where we can make progress.

Isabelle's set theory records nearly 700 theorems. We have discussed the formal development, starting from the ZF axioms, of a calculus of sets, pairs, relations and functions. This is the starting point for a computational logic. The next developments concern general principles for defining recursive data types, including the natural numbers — using, for the first time, the Axiom of Infinity! The companion paper will discuss recursion in all its forms.

Acknowledgements. Philippe Nöel's version of set theory, modified by Martin Coen, was the starting point of the present theory. Tobias Nipkow made great contributions to Isabelle, including the simplifier. David Basin, Matt Kaufman, Brian Monahan and Philippe Noël commented usefully on this work.

References

[1] Peter B. Andrews, Dale A. Miller, Eve L. Cohen, and Frank Pfenning. Automating higher-order logic. In W. W. Bledsoe and D. W. Loveland, editors, *Automated Theorem Proving: After 25 Years*, pages 169–192. American Mathematical Society, 1984.

[2] David Basin and Matt Kaufmann. The Boyer-Moore prover and Nuprl: An experimental comparison. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 89–119. Cambridge University Press, 1991.

[3] Robert Boyer, Ewing Lusk, William McCune, Ross Overbeek, Mark Stickel, and Lawrence Wos. Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning*, 2:287–327, 1986.

[4] Robert S. Boyer and J Strother Moore. *A Computational Logic Handbook.* Academic Press, 1988.

[5] Martin Coen. *Interactive Program Derivation.* PhD thesis, University of Cambridge, 1992.

[6] Michael J. C. Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In G. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153–177. North-Holland, 1986.

[7] Michael J. C. Gordon. HOL: A proof generating system for higher-order logic. In Graham Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer Academic Publishers, 1988.

[8] Paul R. Halmos. *Naive Set Theory.* Van Nostrand, 1960.

[9] Tobias Nipkow. Constructive rewriting. *Computer Journal*, 34:34–41, 1991.

[10] Philippe Noël. Experimenting with Isabelle in ZF set theory. *Journal of Automated Reasoning.* In press.

[11] Bengt Nordström, Kent Petersson, and Jan Smith. *Programming in Martin-Löf's Type Theory. An Introduction.* Oxford, 1990.

[12] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5:363–397, 1989.

[13] Lawrence C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[14] Lawrence C. Paulson. Introduction to Isabelle. Technical report, University of Cambridge Computer Laboratory, 1992.

[15] Lawrence C. Paulson. The Isabelle reference manual. Technical report, University of Cambridge Computer Laboratory, 1992.

[16] Lawrence C. Paulson. Isabelle's object-logics. Technical report, University of Cambridge Computer Laboratory, 1992.

[17] F. J. Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986. Errata, JAR 4 (1988), 235–236.

[18] Art Quaife. Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, 8(1):91–147, 1992.

[19] Herbert John Ryser. *Combinatorial Mathematics.* Mathematical Association of America, 1963.

[20] Mark Saaltink. The EVES library models. Technical Report TR-91-5449-04, ORA Canada, 265 Carling Avanue, Suite 506, Ottawa, Ontario, 1992.

[21] David Schmidt. Natural deduction theorem proving in set theory. Technical Report CSR-142-83, Department of Computer Science, University of Edinburgh, 1983.

[22] Patrick Suppes. *Axiomatic Set Theory.* Dover, 1972.