Proof Pearl: Defining Functions over Finite Sets

Tobias Nipkow[1] and Lawrence C. Paulson[2]

[1] Institut für Informatik, Technische Universität München, Germany
[2] Computer Laboratory, University of Cambridge, England

Abstract. Structural recursion over sets is meaningful only if the result is independent of the order in which the set's elements are enumerated. This paper outlines a theory of function definition for finite sets, based on the fold functionals often used with lists. The fold functional is introduced as a relation, which is then shown to denote a function under certain conditions. Applications include summation and maximum. The theory has been formalized using Isabelle/HOL.

1 Introduction

Finite sets have many applications in interactive proof. Lists are more commonly used, but are a poor substitute for sets: if the order of the list elements really doesn't matter, then the use of lists will introduce meaningless distinctions and pointless complications. In general, of course, sets do not have to be finite. Theorem provers based on higher order logic can easily reason about infinite sets. Finite sets are appropriate for modelling computational phenomena such as message buffering, where the order and possible repetition of messages must be ignored. Finite sets support operations—such as summation, maximum and minimum, cardinality—that are meaningful for infinite sets only in the context of the calculus or other advanced methods.

Fold functionals are a convenient means of defining such operations. For lists, fold functionals are well known [8], but for finite sets, they are problematical. The problem is obvious: fold functionals seem to presuppose an ordering of the elements, when by definition a set is unordered. The solution is to define a fold functional that allows the definition of operations where the order of elements is irrelevant.

- Summations are unaffected by the order in which elements are added.
- The maximum and minimum do not depend on the order in which elements appear.
- Cardinality is unaffected by the order in which elements are counted.

This paper has several objectives. We describe how to the formalize fold functionals for finite sets. While presenting our approach, which we believe provides maximum flexibility for minimal effort, we identify alternatives and outline their merits. No mathematical novelties appear here, just a carefully tuned series of definitions. We use Isabelle/HOL [9], but the approach should be applicable

J. Hurd and T.F. Melham (Eds.): HM 2005, LNCS 3603, pp. 385–396, 2005.

to other theorem provers based on higher-order logic or set theory. We also illustrate the technique of proving a function to be well-defined by showing the corresponding relation to be single valued.

We pay particular attention to the algebraic properties required for iterating a function over a set. It turns out that there are two distinct cases: commutative monoids with a unit (useful for defining summation) and ordered structures (useful for defining minimum). Both require distinct fold functionals and their own theory. In the development of these theories we demonstrate *locales*, a lesser-known Isabelle feature.

As we go along, we compare our approach with the one in HOL4 [7] and PVS [13], both of which provide their own libraries of functions over finite sets. Our basic fold function resembles one formalized for HOL88 by Chou [3].

2 Finite Sets and the Fold Function

We assume there already is a formalization of sets, with standard operations such as comprehension, union and intersection. In higher-order logic, this is trivial by the obvious representation of sets by predicates.

We use standard mathematical notation with a few extensions. Type variables are written $'a$, $'b$, etc., and function types are written using $\Rightarrow$, as in $'a \Rightarrow 'a$. Logical equivalence is denoted by $=$, equality between booleans. The type of sets over a type $'a$ is $'a\ set$. The image of a function over a set, namely $\{f\,x \mid x \in A\}$, is written f ‘ A. Injectivity is written *inj-on f A*, which means the function f is injective when restricted to the set A.

We use two description operators, *SOME* and *THE*. Both denote a value that is specified by a formula. *SOME* is Hilbert's ϵ-operator; it does not require the formula to specify the value uniquely, instead choosing one using the axiom of choice. *THE* is a *definite* description: the specified value must be unique.

Most uses of *SOME* that we have seen can be replaced by *THE*. Sometimes, *SOME* is essential or at least leads to shorter definitions. However, introducing a needless dependence on the axiom of choice is inelegant. Certain formal systems are incompatible with choice [5, Remark 4.6].

2.1 Finite Sets

Finiteness we express by an inductive definition. The empty set is finite, and adding one element to a finite set yields a finite set. The inductive definition defines a predicate *finite* characterizing the finite sets:

$$\mathit{finite}\ \emptyset \qquad \frac{\mathit{finite}\ A}{\mathit{finite}\ (\{a\} \cup A)}$$

In Isabelle, we define the function *insert* and abbreviate the conclusion of the second rule as *finite (insert a A)*. This function brings out the analogy between finite sets built by $\emptyset$ and *insert* with lists built by [] and *Cons*. We do not introduce a separate type of finite sets, which might be desirable in systems that offer predicate subtyping.

Isabelle's inductive definition package also generates an induction rule similar to structural induction for lists. Familiar properties are easily proved by induction. For example, the union of finitely many finite sets is itself finite.

Traditionally, a set is finite if it can be put into one-to-one correspondence with the set of natural numbers less than some n. Another traditional definition says A is finite if every injection from A to A is also a surjection. Compared with the inductive definition of finiteness, which allows the standard results to be proved easily, such definitions are inconvenient. Their only advantage is that they do not presuppose the concept of inductive definition.

If we did not start from an existing type of sets which includes the infinite ones, we could define the type of finite sets as a quotient type of a free algebra [11]. There are two standard approaches.

- empty set, singleton set and union, modulo associativity, commutativity and idempotency of union
- empty set and insert, modulo *left-commutativity* and *left-idempotency*:

$$\begin{gathered} \mathit{insert}\ a\ (\mathit{insert}\ b\ A) = \mathit{insert}\ b\ (\mathit{insert}\ a\ A) \\ \mathit{insert}\ a\ (\mathit{insert}\ a\ A) = \mathit{insert}\ a\ A \end{gathered}$$

Below we will refer to them as the $(\emptyset,\{-\},\cup)$-algebra and the $(\emptyset,\mathit{insert})$-algebra.

2.2 Which Fold Function?

Our main interest is defining functions over finite sets by recursion. For lists, this is trivial, but for sets, we must ensure that the order of the elements is irrelevant. The cardinality of a set is a key function that might be defined recursively, though it turns out to be essential in the development of recursion in the first place.

We seek a function *fold* of type $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'b\ \mathit{set} \Rightarrow 'a$. It should satisfy the equation

$$\mathit{fold}\ f\ g\ z\ \{x_1, ..., x_n\} = f\ (g\ x_1)\ (\ldots\ (f\ (g\ x_n)\ z)\ldots)$$

if f is associative and commutative (AC). The function g is applied to each of $\{x_1, ..., x_n\}$, then z is thrown in, and the resulting values —which do *not* have to be distinct—are combined using f. This fold function is also well known from functional data base query languages, for example Machiavelli [10].

If e is a unit element for f, i.e. $f\ x\ e = x$, then *fold* satisfies the recursion equations

$$\begin{gathered} \mathit{fold}\ f\ g\ e\ \emptyset = e \\ \mathit{fold}\ f\ g\ e\ \{a\} = g\ a \\ \mathit{fold}\ f\ g\ e\ (A \cup B) = f\ (\mathit{fold}\ f\ g\ e\ A)\ (\mathit{fold}\ f\ g\ e\ B) \end{gathered}$$

if A and B are finite disjoint sets. Hence *fold* corresponds to the $(\emptyset,\{-\},\cup)$-algebra view of finite sets.

Can the argument g be eliminated? The application of g to the set elements cannot be replaced by a separate use of the image operator because the resulting collection of values must not be regarded as a set. We could combine f and g

into one function, namely $\lambda x\ y.\ f\ (g\ x)\ y$. That approach, followed in HOL4, resembles the treatment of fold for lists, i.e. it takes the ($\emptyset$,*insert*)-algebra view of finite sets. It can be shown that the two fold functions are interdefinable [2]. Our treatment has the advantage that it only involves standard algebraic properties like associativity and commutativity.

What happens if there is no unit e for f? As an example, consider a naive definition of the minimum of a set of natural numbers: $Min \equiv fold\ min\ id\ 0$ where *min* is the binary minimum (which is AC, but has no unit). The sad consequence of this definition is that *Min* always returns zero. As a matter of fact, *fold* will not allow us to define the minimum of a set over any type that lacks a greatest element: a unit for *min*. Similarly, the maximum of a set can be defined using *fold* only if the type has a least element; for the natural numbers, it is correct to define $Max \equiv fold\ max\ id\ 0$. We treat the case of a missing unit element separately in §4.

2.3 Defining the Fold Function

We do not define *fold* directly. Instead, we inductively define its graph: its input/output relation. After proving that this relation is deterministic, we use it to define *fold*. The relation is a constant *foldSet* of type

$$('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a \Rightarrow ('b\ set \times 'a)\ set.$$

Its inductive definition has two introduction rules. The first, intuitively, states that when applied to the empty set, $fold\ f\ g\ z$ should return z.

$$(\emptyset,\ z) \in foldSet\ f\ g\ z$$

The second rule says that if the "fold" of A can yield y, then the "fold" of $\{x\} \cup A$ can yield $f\ (g\ x)\ y$, for any $x \notin A$.

$$\frac{x \notin A \qquad (A,\ y) \in foldSet\ f\ g\ z}{(\{x\} \cup A,\ f\ (g\ x)\ y) \in foldSet\ f\ g\ z}$$

Proving that the "fold" of A can yield only one value is complicated because the elements of A might be inserted in any order. Our proof (see below) is by induction on the cardinality of A. Once we have proved that *foldSet* corresponds to a function, we can define the "fold" of A to be the unique x determined by $foldSet\ f\ g\ z$:

$$fold\ f\ g\ z\ A \equiv THE\ x.\ (A,\ x) \in foldSet\ f\ g\ z$$

This step requires the unique description operator, but not the axiom of choice.

3 A Fold Function for Finite Sets

All treatments of fold that we are aware of require some notion of finite cardinality. Various definitions are possible.

1. The traditional approach refers to a bijection to an initial segment of the natural numbers: $card\ A \equiv LEAST\ n.\ \exists f.\ A = f\ `\ \{m \mid m < n\}$
2. HOL4 defines cardinality as a relation of pairs $(A,\ n)$. This graph is proved to be deterministic and turned into a function using a description.
3. The approach we adopt below refers implicitly to the traditional definition of finiteness, via the theorem $finite\ A = (\exists\, n\ f.\ A = f\ `\ \{i \mid i < n\})$.

Alternative 2 above resembles the inductive definition of *fold*. Whichever alternative is chosen, we should only prove enough results about cardinality to allow the definition of *fold*: many lemmas about cardinality are instances of more general lemmas about set summation and can be obtained easily once that concept is defined with the help of *fold*. We come back to this point in §3.5.

As stated in the previous section, the relation *foldSet* is defined inductively. In order to turn *foldSet* into a function, we must show *determinacy*: for each A there is at most one y such that $(A,\ y) \in foldSet\ f\ g\ x$. However, this is not true for arbitrary f: the function must be AC.

3.1 Commutative Monoids

An Isabelle *locale* is essentially a detached proof context, comprising variables and assumptions that are temporarily treated like constants and axioms [1]. When proving theorems within a locale, there is no need to repeat the assumptions nor to discharge them when appealing to other theorems of that locale.

Here, a locale conveniently packages the function f with its associative and commutative laws. Locale *ACf* even gives f an infix syntax $\cdot$ to improve clarity:

locale *ACf* =
  fixes $f :: 'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\cdot$ *70*)
  assumes *commute*: $x \cdot y = y \cdot x$
  and *assoc*: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

Some of our proofs require a unit element for f. Locale *ACe* extends *ACf* with such an element, called e:

locale *ACe* = *ACf* +
  fixes $e :: 'a$
  assumes *ident*: $x \cdot e = x$

For clarity, we write $\cdot$ and e only in the context of the corresponding locale. Outside the locale, where the values are unconstrained, we write f and z. In the locale, references to f as a function appear as $(\cdot)$, a notation that emphasizes the connection with the infix syntax $x \cdot y$. An example of this notation in context is $foldSet\ (\cdot)\ g\ x$.

3.2 From Relation *foldSet* to Function *fold*

The determinacy of *foldSet* is easily stated:

If $(A,\ x) \in foldSet\ (\cdot)\ g\ z$ and $(A,\ y) \in foldSet\ (\cdot)\ g\ z$ then $y = x$.

For induction, we insert two conditions, expressing that A has cardinality n:

If $A = h\ `\ \{i \mid i < n\}$ and *inj-on* $h\ \{i \mid i < n\}$ and $(A, x) \in \mathit{foldSet}\ (\cdot)\ g\ z$ and $(A, y) \in \mathit{foldSet}\ (\cdot)\ g\ z$ then $y = x$.

The proof is conducted within locale *ACf*, making available the assumption that $\cdot$ is associative and commutative. We use complete induction on n, obtaining as induction hypothesis that the theorem statement holds for all m less than n.

We next analyse the assumption $(A, y) \in \mathit{foldSet}\ (\cdot)\ g\ z$ by cases on the definition of *foldSet*. If $A = \emptyset$ then both x and y are equal to z. Otherwise A is non-empty. We must consider two potential evaluations of fold:

1. $A = \{b\} \cup B$ and $x = g\ b \cdot u$ and $(B, u) \in \mathit{foldSet}\ (\cdot)\ g\ z$, where $b \notin B$
2. $A = \{c\} \cup C$ and $y = g\ c \cdot v$ and $(C, v) \in \mathit{foldSet}\ (\cdot)\ g\ z$, where $c \notin C$

If $b = c$ then also $B = C$, whence $u = v$ by the induction hypothesis. We thus find that x and y equal $g\ b \cdot u$.

The only remaining case is if $b \neq c$. Here, we define the set $D = B - \{c\}$. Trivially $B = \{c\} \cup D$, but we can also show $C = \{b\} \cup D$ by subtracting c from both sides of the set equation $\{b\} \cup B = \{c\} \cup C$. Because the set D is of smaller cardinality, the induction hypothesis tells us that $(D, w) \in \mathit{foldSet}\ f\ g\ z$ for a unique w. Returning to the two potential evaluations of fold, we find

1. $A = \{b, c\} \cup D$ and $x = g\ b \cdot (g\ c \cdot w)$
2. $A = \{c, b\} \cup D$ and $y = g\ c \cdot (g\ b \cdot w)$.

Associativity and commutativity of $(\cdot)$ imply $x = y$, which completes the proof. □

If a theory of finite cardinality is already available, we can replace the formulas $A = h\ `\ \{i \mid i < n\}$ and *inj-on* $h\ \{i \mid i < n\}$ above by *finite* A. The proof would then require a complete induction on the cardinality of A. The only properies of cardinality needed are the obvious ones: $\mathit{card}\ \emptyset = 0$ and

$$\mathit{card}\ (\{a\} \cup A) = (\mathsf{if}\ a \in A\ \mathsf{then}\ \mathit{card}\ A\ \mathsf{else}\ \mathit{card}\ A + 1).$$

Our experience is that defining cardinality separately requires much work and yields only a small simplification in the formalization of the proof above. Fold will give us cardinality for free.

Now that we have shown that *foldSet* is the graph of a function, we can easily derive the recursion rule for *fold* where A must be finite and $x \notin A$:

$$\mathit{fold}\ (\cdot)\ g\ z\ (\{x\} \cup A) = g\ x \cdot \mathit{fold}\ (\cdot)\ g\ z\ A \tag{1}$$

The base case $\mathit{fold}\ f\ g\ z\ \emptyset = z$ follows directly from the definitions. Once we have these two equations, we can forget about the original definition of *fold*.

3.3 An Alternative: Defining Fold by Choice and Recursion

An alternative definition of *fold* is by well-founded recursion over the cardinality of the set, where $\mathit{pick}\ A \equiv \mathit{SOME}\ a.\ a \in A$ and $\mathit{rest}\ A \equiv A - \mathit{pick}\ A$:

$\mathit{fold}\ f\ g\ x\ A = (\mathsf{if}\ A = \emptyset\ \mathsf{then}\ x\ \mathsf{else}\ \mathit{fold}\ f\ g\ (\mathit{pick}\ A)\ (\mathit{rest}\ A))$

This version, used in PVS and HOL4, is appealingly concise. In PVS the type of *fold* is restricted to finite sets, whereas in HOL the recursion equation is subject to the condition *finite A*. A comparison of the Isabelle and HOL4 proof scripts leading to equation (1) suggests that the use of the axiom of choice yields no reduction in the proof effort. Our approach, which introduces no dependence on the axiom of choice, provides more flexibility at no additional cost.

3.4 Further Properties of Fold

Here is a selection of equations that we have formally proved about the function *fold*. They are all subject to the condition that the sets are finite. Most of the proofs are trivial inductions over the finite set A.

$f\ x\ (fold\ f\ g\ z\ A) = fold\ f\ g\ (f\ x\ z)\ A$
$fold\ f\ g\ (fold\ f\ g\ z\ B)\ A = fold\ f\ g\ (fold\ f\ g\ z\ (A \cap B))\ (A \cup B)$
$fold\ f\ g\ z\ (h\ `\ A) = fold\ f\ (g \circ h)\ z\ A$

The following theorems is proved in locale ACe:

$$fold\ (\cdot)\ g\ e\ A \cdot fold\ (\cdot)\ g\ e\ B = fold\ (\cdot)\ g\ e\ (A \cup B) \cdot fold\ (\cdot)\ g\ e\ (A \cap B) \qquad (2)$$

These samples from the fold library should suffice.

3.5 Applications

Our experience is that users seldom invoke *fold* directly. The great majority of references to *fold* are via functions defined using it. Summation, which is the most important of these, sums a given function over an index set. Also useful is an analogous operator for products. Cardinality is defined in terms of summation.

So far we have implicitly worked in the context of a fixed but arbitrary commutative semigroups or monoid, i.e. the locales ACf or ACe. Now we replace $\cdot$ by addition or multiplication and e by *0* or *1*. This is fine as those are commutative monoids. In Isabelle it means instantiating the locales ACf or ACe and proving their assumptions; the details are discussed elsewhere [1].

Sums Over a Set. We work in a flexible formalization of arithmetic, defined using axiomatic type classes [12]. A general theory of commutative monoids specifies that + is an AC operator with unit element 0. The resulting concept of summation is applicable to integers, rationals, reals, matrices and even multisets.

The sum of the function f over the set A is defined in terms of *fold*.

$setsum\ f\ A \equiv$ *if finite A then fold* $(+)\ f\ 0\ A$ *else 0*

The sum is defined to be zero if A is infinite; by case analysis on *finite A*, many theorems about *setsum* can be proved without finiteness assumptions. (The analogous if-then definition of *fold* would simplify only a few theorems.) This summation operator inherits the theorems about *fold* shown above.

Products over a set are defined analogously.

Syntactic Sugar. We have attached more conventional concrete syntax to various forms of *setsum*. For a start *setsum* $(\lambda x.\ e)\ A$ can be written (and is always printed) as $\sum x{\in}A.\ e$. Instead of $\sum x{\in}\{x.\ P\}.\ e$ we have the shorter $\sum x \mid P.\ e$. The special form $\sum x{\in}A.\ x$ abbreviates to $\sum A$.

Cardinality of a Finite Set. As remarked above, the summation operator and the theorems proved about it are applicable to all types that belong to the class of commutative monoids. Among these is *nat*, the type of the natural numbers. The cardinality of a finite set can be expressed as a summation:

$card\ A \equiv \sum x{\in}A.\ 1$

Note that the cardinality of an infinite set is zero with this definition. The usual properties of cardinality are instances of those for summations.

4 A Fold Function for Non-empty Sets

Some functions on finite sets, such as *Max*, require their argument to be non-empty. The algebraic reason is that the result type does not have a unit element e. This phenomenon is well-known from functional programming with lists, where typically two fold functions are available. We do the same here and define a second fold function *fold1* of type $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ set \Rightarrow 'a$ such that

$fold1\ (\cdot)\ \{x_1, ..., x_n\} = x_1 \cdot \ldots \cdot x_n$

if $(\cdot)$ is associative and commutative.

Unlike *fold*, the function *fold1* does not apply some g to all x_i. In all our examples, the operator $(\cdot)$ is idempotent, when g can be mapped over the set first. For *fold*, it is easy to show that if $(\cdot)$ is idempotent then g becomes redundant:

$finite\ A \Longrightarrow fold\ (\cdot)\ g\ z\ A = fold\ (\cdot)\ id\ z\ (g\ `\ A)$

The Isabelle definition of *fold1* again avoids the axiom of choice and proceeds as for *fold*: a relation *fold1Set* is defined inductively. We avoid recursion and reduce *fold1Set* to *foldSet*.

$$\frac{(A, x) \in foldSet\ f\ id\ a \qquad a \notin A}{(\{a\} \cup A, x) \in fold1Set\ f}$$

Again, our plan is to convert this relation into a function:

$fold1\ f\ A \equiv THE\ x.\ (A, x) \in fold1Set\ f$

A surprise: this does *not* require proving determinacy of *fold1Set*! The equation for the base case is easy to show: $fold1\ f\ \{a\} = a$. Harder to derive is the recursion rule, where A must be finite and non-empty and $a \notin A$:

$$fold1\ (\cdot)\ (\{a\} \cup A) = a \cdot fold1\ (\cdot)\ A \tag{3}$$

Our proof requires two lemmas to allow us to change *foldSet*'s third argument. Both lemmas are proved by induction on the derivation of their first premise.

$$\frac{(A,\ y) \in \mathit{foldSet}\ (\cdot)\ \mathit{id}\ b \qquad b \notin A}{(\{b\} \cup A,\ z \cdot y) \in \mathit{foldSet}\ (\cdot)\ \mathit{id}\ z}$$

$$\frac{(A,\ x) \in \mathit{foldSet}\ (\cdot)\ \mathit{id}\ b \qquad a \in A \qquad b \notin A}{(\{b\} \cup (A - \{a\}),\ x) \in \mathit{foldSet}\ (\cdot)\ \mathit{id}\ a}$$

From these two lemmas, we can prove an equation relating *fold1* to *fold*

$$\mathit{fold1}\ (\cdot)\ (\{a\} \cup A) = \mathit{fold}\ (\cdot)\ \mathit{id}\ a\ A$$

where again A must be finite and $a \notin A$. The recursion rule (3) follows easily. If $(\cdot)$ is idempotent, then $a \notin A$ can be dropped. The same holds for *fold*, but it is less useful there, because few applications of *fold* involve idempotent operators.

4.1 Properties

In the sequel, all sets are implicitly assumed to be finite and non-empty.

There are fewer general properties of *fold1* than of *fold* because *fold1* has fewer parameters. Of the *fold* lemmas in §3.4, only a single one still makes sense here: provided $A \cap B = \emptyset$, *fold1* distributes over union.

$$\mathit{fold1}\ (\cdot)\ (A \cup B) = \mathit{fold1}\ (\cdot)\ A \cdot \mathit{fold1}\ (\cdot)\ B$$

For *fold*, this distributive law is the corollary of the more general lemma (2), which does not hold for *fold1*.

If $(\cdot)$ is idempotent as well, the premise $A \cap B = \emptyset$ in the distributive law can be dropped. In fact, idempotence of $(\cdot)$ makes *fold1* come into its own.

We will now examine properties of *fold1* in various ordered structures (for details see the literature [4]). These structures are again formalized by locales in Isabelle, but our presentation will stay on an abstract mathematical level.

4.2 Semilattices

In this subsection, we assume $(\cdot)$ is not just AC but also *idempotent*: $x \cdot x = x$. This means we are in a semilattice. To obtain the order-theoretic view, we define the symbol $\sqsubseteq$ by

$$(x \sqsubseteq y) = (x \cdot y = x).$$

The semi-lattice law $(x \sqsubseteq y \cdot z) = (x \sqsubseteq y \wedge x \sqsubseteq z)$ has a nice generalization in terms of *fold1*:

$$(x \sqsubseteq \mathit{fold1}\ (\cdot)\ A) = (\forall\, a{\in}A.\ x \sqsubseteq a)$$

The dual property $(x \cdot y \sqsubseteq z) = (x \sqsubseteq z \vee y \sqsubseteq z)$ holds iff the ordering $\sqsubseteq$ is linear. Then, it can be generalized to

$$(\mathit{fold1}\ (\cdot)\ A \sqsubseteq x) = (\exists\, a{\in}A.\ a \sqsubseteq x).$$

Note that only the left-to-right direction of this equivalence requires linearity. The other direction is valid in arbitrary semilattices:

$$a \in A \Longrightarrow \mathit{fold1}\ (\cdot)\ A \sqsubseteq a, \tag{4}$$

4.3 Lattices

Frequently, the semilattice is in fact a lattice: there is not just an infimum ($(\cdot)$ above) but also a supremum, which we write $\sqcap$ and $\sqcup$. With the help of *fold1*, we can define the standard extension of $\sqcap$ and $\sqcup$ to finite sets:

$$\bigsqcap A \equiv \mathit{fold1}\ (\sqcap)\ A \qquad \bigsqcup A \equiv \mathit{fold1}\ (\sqcup)\ A$$

We inherit the semilattice laws and can derive new ones from them. For example, we obtain $\bigsqcap A \sqsubseteq \bigsqcup A$ from the instances of (4) for the two semilattices $\sqcap$ and $\sqcup$.

In case of a distributive lattice, distributivity propagates from the binary to the n-ary operations. Here are two examples:

$$x \sqcup \bigsqcap A = \bigsqcap \{x \sqcup a \mid a \in A\} \qquad \bigsqcap A \sqcup \bigsqcap B = \bigsqcap \{a \sqcup b \mid a \in A \wedge b \in B\}$$

4.4 Applications

The most direct applications of *fold1* are minimum and maximum because (as noted in §2.2) many types lack a least or greatest element, which means *fold* is inappropriate. In Isabelle, we can define *Min* and *Max* as follows:

$$\mathit{Min} \equiv \mathit{fold1}\ \mathit{min} \qquad \mathit{Max} \equiv \mathit{fold1}\ \mathit{max}$$

Here, *min* and *max* are overloaded functions available on any type of class *ord*, which means the type must define an ordering $\leq$. Hence, *Min* and *Max* have type $'a\ \mathit{set} \Rightarrow 'a$, where $'a$ is of class *ord*.

We can now inherit all of the *fold1* properties described above because *Min* and *Max* form a distributive lattice. After instantiating the corresponding locales we obain, for example, the distributive law

$$\mathit{max}\ (\mathit{Min}\ A)\ (\mathit{Min}\ B) = \mathit{Min}\ \{\mathit{max}\ a\ b \mid a \in A \wedge b \in B\}$$

Functions *Min* and *Max* and their properties are now available, for example, on all numeric types (except the complex numbers) as they are linearly ordered.

In a similar manner, we can define the greatest common divisor and the least common multiple of a set of natural numbers: $\mathit{Gcd} \equiv \mathit{fold1}\ \mathit{gcd}$ and $\mathit{Lcm} \equiv \mathit{fold1}\ \mathit{lcm}$, where *gcd* and *lcm* are the binary versions. Functions *gcd* and *lcm* also form a distributive lattice, where the ordering is divisibility. They even form a complete lattice.

Finally, we consider the longest common prefix (*lcp*) of two lists:

$$\begin{aligned} \mathit{lcp}\ []\ \mathit{xs} &= [] \\ \mathit{lcp}\ \mathit{xs}\ [] &= [] \\ \mathit{lcp}\ (x\ \#\ \mathit{xs})\ (y\ \#\ \mathit{ys}) &= (\mathsf{if}\ x = y\ \mathsf{then}\ x\ \#\ \mathit{lcp}\ \mathit{xs}\ \mathit{ys}\ \mathsf{else}\ []) \end{aligned}$$

where $\#$ is the list *Cons*. Of course, the corresponding ordering is the prefix ordering. This only yields a lower semilattice: there is no greatest element and no supremum. Thus we only define $\mathit{Lcp} \equiv \mathit{fold1}\ \mathit{lcp}$.

4.5 Alternative Definitions

As in §3.3, we can use the axiom of choice to define *fold1* by the obvious recursion on the cardinality. Again, it is simpler to define *fold1* in terms of *fold*:

$$fold1\ f\ A \equiv fold\ f\ id\ (pick\ A)\ (rest\ A)$$

The main advantage of this definition is that the recursion rule (3) can now be proved by a few case distinctions, a few properties of *pick*, and equational reasoning alone, assuming we already know

$$a \in A \Longrightarrow fold\ (\cdot)\ g\ z\ A = g\ a \cdot fold\ (\cdot)\ g\ z\ (A - \{a\})$$
$$x \cdot fold\ (\cdot)\ g\ z\ A = fold\ (\cdot)\ g\ (x \cdot z)\ A$$

both of which are natural lemmas for *fold*. This proof of (3) has the advantage over the one via *fold1Set* that it does not require any special purpose lemmas.

Neither PVS nor HOL4 provide an analogue of *fold1*. Both systems define the minimum and maximum of a set directly. Other functions like the above *Gcd* and *Lcp* would need to be defined separately.

HOL4 defines *Min* and *Max* only for sets of natural numbers. An advantage of the HOL4 approach is that *Min* also works for infinite sets. Folding does not make sense for infinite sets unless we introduce some notion of limit.

Let us compare the different approaches and assume we are interested in finite sets only. Then *fold1* has the advantage of generality over special purpose *Min* and *Max* definitions. The shortest definitions and proofs are obtained by defining *fold1* with the help of *fold* and choice. The inductive definition is a bit lengthier but not significantly so, and avoids choice. But even if one is just interested in *Min* and *Max*, their definition by description is not ideal. It is simpler by far to derive the characteristic properties of *Min* and *Max* from the recursion equations than the other way around. This can be seen by comparing the Isabelle and HOL4 proofs.

5 Conclusions

Recursive function definitions over finite sets are not difficult to justify. The mathematics is simple. We have taken pains to ensure that the machine formalization is simple too, while avoiding any dependence on the axiom of choice. The applications we have discussed are cardinality, sum and product over sets. For the case of non-empty sets, we have discussed the maximum and minimum operators.

One question we have not discussed at all is definability. It is easily seen that not every function on finite sets is definable by means of fold: if $F\ A \equiv card\ A \leq 1$ were definable as $F \equiv fold\ (\cdot)\ g\ e$ for suitable $(\cdot)$, g and e, then it would follow that $e = F\ \emptyset = True$, $g\ x \cdot e = F\ \{x\} = True$ and hence $False = F\{x,y\} = g\ x \cdot g\ y \cdot e = g\ x \cdot e = True$. On the other hand *fold* can trivially define any homomorphism from the finite sets viewed as a $(\emptyset,\{-\},\cup)$-algebra into a $(e,g,\cdot)$-algebra: $F \equiv fold\ (\cdot)\ g\ e$. But being a homomorphism this implies that $\cdot$

must satisfy all union laws, in particular idempotence: $F\ A = F(A \cup A) = F\ A \cdot F\ A$. Hence set sum and product are not homomorphisms but still defineable. This shows that the definability question is outside the scope of this paper and requires a separate study analogous to the work of Gibbons *et al.* [6] for lists.

References

1. C. Ballarin. Locales and locale expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *International Workshop TYPES 2003*, LNCS 3085, pages 34–50. Springer, 2004.
2. V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with sets/bags/lists. In J. L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *Automata, Languages and Programming (ICALP91)*, volume 510 of *LNCS*, pages 60–75. Springer, 1991.
3. C.-T. Chou. Generalizing an associative and commutative operation with identity to finite sets. ftp://ftp.cl.cam.ac.uk/hvg/hol88/contrib/aci/, 1992. HOL88 formal development.
4. B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
5. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.
6. J. Gibbons, G. Hutton, and T. Altenkirch. When is a Function a Fold or an Unfold? In *Proc. 4th International Workshop on Coalgebraic Methods in Computer Science*, volume 44.1 of *Electronic Notes in Theoretical Computer Science*, 2001.
7. The HOL4 Theorem Prover. http://hol.sf.net.
8. G. Hutton. A tutorial on the universality and expressiveness of fold. *Journal of Functional Programming*, 9(4):355–372, July 1999.
9. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002. LNCS Tutorial 2283.
10. A. Ohori, P. Buneman, and V. Tannen. Database programming in machiavelli - a polymorphic language with static type inference. In J. Clifford, B. Lindsay, and D. Maier, editors, *Proc. 1989 ACM SIGMOD Intl. Conf. Management of Data*, pages 46–57. ACM Press, 1989.
11. L. C. Paulson. Defining functions on equivalence classes. *ACM Transactions on Computational Logic*. in press.
12. L. C. Paulson. Organizing numerical theories using axiomatic type classes. *Journal of Automated Reasoning*, 2005. in press.
13. PVS Specification and Verification System. http://pvs.csl.sri.com/.