A machine assisted formalization of pointfree topology in type theory

Jan Cederquist
Department of Computing Science
University of Göteborg
S-412 96 Göteborg, Sweden
email: ceder@cs.chalmers.se

Abstract

We will present a formalization of pointfree topology in Martin-Löf's type theory. A notion of point will be introduced and we will show that the points of a Scott topology form a Scott domain. This work follows closely the intuitionistic approach to pointfree topology and domain theory, developed mainly by Martin-Löf and Sambin. The important difference is that the definitions and proofs are machine checked by the proof assistant ALF.

Contents

1 Introduction

The traditional motivation for topology relies on abstracting first from Euclidean spaces to metric spaces, and then abstracting out certain properties of their open sets.

A topology ΩX for a set X is a family of subsets of X which is closed under finite intersection and arbitrary union. X is the space of the topology and $\langle X, \Omega X\rangle$ is a topological space. The elements in X are called points and the sets in ΩX are called open sets. For the development of general topology see for instance Kelley [8].

In pointfree topology (locale theory), Johnstone [7], one considers the open sets, and not the points, as primitive entities and studies those properties of a topological space that can be expressed without any mention of points. By abstracting from the fact that open sets are subsets of points one only looks at the algebraic structure, called a *frame*, that the open sets form.

A frame is a partially ordered set A with two operations *meet* $\wedge$ and *join* $\vee$, operating on subsets of A, corresponding to intersection and union, respectively. *Meet* gives to each finite subset the infimum and *join* gives to each subset the supremum. In particular, A contains two elements $true$ and $false$ which correspond to empty meet and join, respectively. Binary meets must also distribute over join. We call the elements in a frame for *opens*.

On the computer science side the motivation for topology relies on connections to domain theory. Scott [14] has showed that by describing only the elements that contain a finite amount of information, the computational content of a domain can be described topologically. This emphasis makes the open sets independent of the points of the topological space, leading to pointfree topology.

Given a frame, points can be defined uniquely from the opens as *completely prime filters* [16]. As an example, take the special case when the opens are open sets (and $true$, $\wedge$, $\vee$ and $\leq$ are the whole space, $\bigcap$, $\bigcup$ and $\subseteq$, respectively). A completely prime filter F then corresponds to the points in the intersection of the open sets in F. In other words, given a point x the corresponding completely prime filter is the set of all open sets containing x. Let $\langle A, \wedge, \vee\rangle$ be a frame and let $F \subseteq A$ be upper closed, that is if $a \in F$ and $a \leq b$ then $b \in F$, then

F is a *filter* iff it is closed under finite meets:
$true \in F$ and if $a, b \in F$ then $a \wedge b \in F$,

a filter F is *completely prime* iff it is inaccessible by joins:
if $S \subseteq A$ and $\bigvee S \in F$ then $s \in F$ for some $s \in S$.

Another motivation for pointfree topology is constructiveness; sometimes the use of pointfree topology makes it possible to replace non constructive reasoning using the axiom of choice by constructive proofs, see for instance Coquand [2, 4].

This work is a machine assisted formalization, in type theory [11], of (a part of) the intuitionistic approach to pointfree topology and domain theory, developed by Martin-Löf [10], Sambin [12], and by Sambin, Valentini and Virgili in [13]. All definitions and proofs are checked by the proof assistant ALF [9]. In [12, 13] the constructivity is guaranteed by adopting Martin-Löf's type theory, but in this paper we will by type theory mean the formalization in ALF. We will prove that this formalization really defines a frame, where the opens are defined as equivalence classes of subsets. A closure operator will be defined and we will prove that each equivalence class contains exactly one closed subset. As a concrete example of a pointfree topology we will look at the neighbourhoods of the natural numbers. A notion of point equivalent to completely prime filter will be introduced. Then we will look at a pointfree version of *Scott topology*, and show that the points of this topology form a *Scott domain*.

Another formalization of constructive domain theory, in ALF, is presented in Hedberg[6]. Hedberg has implemented a cartesian closed category of semilattices and approximable mappings.

In Martin-Löf's type theory, which is implemented in ALF, there are two basic levels: types and sets. The sets are inductively defined and correspond to what is usually called types in a programming language. The types are formed by the type `Set`, the types of elements of sets in `Set`, and function types. In type theory propositions are identified with sets and proofs of propositions are identified with elements of sets; in order to prove that a proposition is true we need to find an element in the corresponding set. Three different forms of definitions (apart from definitions of contexts and substitutions, which will be explained later) will be used in this paper:

1. Inductive definitions of sets or families of sets, which consist of a formation rule and introduction rules prescribing how its canonical elements are formed.

2. Explicit definitions, which are names for well typed expressions.

3. Implicit definitions, which provide the possibility of defining functions using pattern matching [3]. These may be recursive.

All type theory expressions will be written in ALF-syntax and in typewriter font. For example, the set of natural numbers is inductively defined by

```
N : Set
  zero : N
  succ : (n:N)N
```

Addition can then be implicitly defined, using pattern matching:

```
add : (n:N;m:N)N
  add(zero,m) = m
  add(succ(n1),m) = succ(add(n1,m))
```

And a function that doubles its argument can be explicitly defined:

```
double = [n]add(n,n) : (n:N)N
```

In this paper, we will often refer to appendices about proofs. However, most proof terms are too long for a comprehensible presentation, so we have decided to omit many of them entirely and only present their types. Instead all proofs can be obtained by ftp; ftp.cs.chalmers.se: /pub/users/ceder/formtop/$*$.

2 Formalization of pointfree topology in ALF

2.1 Formal topology

Following Sambin [12, 13], a structure $\langle S, \wedge, 1, \lhd, pos\rangle$ is called a *formal topology* if it satisfies the following requirements:

1. S is a formal base, that is, a set with the binary operation $\wedge$ and element 1 such that $\langle S, \wedge, 1\rangle$ forms a meet semilattice (that is, an algebra with a unit element and a binary operator satisfying commutativity, associativity, unit law and idempotence). The elements in S are called formal basic neighbourhoods.

2. $\lhd$ is a covering relation, that is, a relation between elements of S and subsets of S which for arbitrary $a, b \in S$ and $U, V \subseteq S$ satisfies:

$$\frac{a \in U}{a \lhd U}\ \text{reflexivity}$$

$$\frac{a \lhd U \quad (\forall b \in U)(b \lhd V)}{a \lhd V}\ \text{transitivity}$$

$$\frac{a \lhd U}{a \wedge b \lhd U}\ \wedge\text{-left1}$$

$$\frac{a \lhd U \quad a \lhd V}{a \lhd \{b \wedge c : b \in U, c \in V\}}\ \wedge\text{-right}$$

3. *pos* is a consistency predicate, that is, a predicate on the elements of S which for arbitrary $a \in S$ and $U \subseteq S$ satisfies:

$$\frac{pos(a) \quad a \lhd U}{(\exists b \in U)\ pos(b)}\ \text{monotonicity}$$

$$\frac{pos(a) \to a \lhd U}{a \lhd U}\ \text{positivity}$$

If we extend $\wedge$, $\lhd$ and *pos* for arbitrary $U, V \subseteq S$ by the definitions

$$U \bigwedge V \equiv \{a \wedge b : a \in U, b \in V\}$$

$$U \lhd V \equiv (\forall a \in U)(a \lhd V)$$

$$POS(U) \equiv (\exists a \in U) pos(a)$$

then transitivity, $\wedge$-right, and monotonicity can be written

$$\frac{a \lhd U \quad U \lhd V}{a \lhd V}\ \text{transitivity}$$

$$\frac{a \lhd U \quad a \lhd V}{a \lhd U \bigwedge V}\ \wedge\text{-right}$$

$$\frac{pos(a) \quad a \lhd U}{POS(U)}\ \text{monotonicity}$$

which are also closer to the forthcoming definitions in type theory. (Observe the introduction of the new symbols $\bigwedge$, $\lhd$ and POS. The reason not to let $\wedge$, $\lhd$ and *pos* be overloaded is that all of them will be defined in ALF and ALF does not support overloading.)

In order to define $\langle S, \wedge, 1\rangle$ to be a semilattice, an ordering or equality between the elements in S is needed. To avoid that, one can notice that if (2) in the definition above holds then $\langle S, \wedge, 1\rangle$ is a semilattice iff for arbitrary $a \in S$ and $U \subseteq S$

$$\frac{b \lhd U}{a \wedge b \lhd U}\ \wedge\text{-left2}$$

and

$$\frac{}{a \lhd \{1\}}\ \wedge\text{-1}$$

hold.

Proof: First, assume that $\langle S, \wedge, 1\rangle$ is a semilattice with equality $=_S$, $\wedge$-left2 then follows from $\wedge$-left1 and commutativity of $\wedge$, and $\wedge$-1 is proved by

$$\frac{\dfrac{\dfrac{\dfrac{}{1 \in \{1\}}}{1 \lhd \{1\}}\ \text{reflexivity}}{a \wedge 1 \lhd \{1\}}\ \wedge\text{-left2} \qquad \dfrac{}{a \wedge 1 =_S a}\ \text{unit}}{a \lhd \{1\}}\ \lhd\ \textit{must respect}\ =_S$$

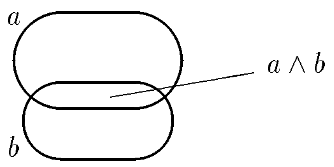
Second, if $\wedge$-left2 and $\wedge$-1 hold then, then we can define an equality between elements in S such that two element are equal if they are covered by each other's singleton sets. Commutativity, associativity, unit law and idempotence for $\wedge$, with respect to that equality are then easily proved; so $\langle S, \wedge, 1\rangle$ form a semilattice with 1 as top element (for more details see appendix D, that $\langle S, \wedge, 1\rangle$ form a semilattice is proved in ALF after the notion of formal topology is defined in type theory).

By exchanging the requirement that $\langle S, \wedge, 1\rangle$ should form a semilattice for the two new rules, we get a definition which is equivalent to the standard definition of formal topology. The reason for this exchange is that it makes the formalization shorter; it is easier to state the new rules than to define a semilattice.

In the definition of formal topology, a subset of S is a propositional function with argument ranging over S. For instance, a is considered as an element in U iff $a \in S$ and $U(a)$ holds. In section 2.3 there is a little theory of these subsets.

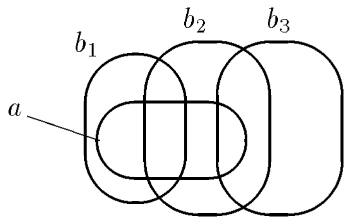
2.2 Explanations of the definition of formal topology

We can think of the elements of S as containing information represented by regions, in such a way that a neighbourhood corresponding to a subregion of another is more informative (it contains more specific information). By $a \wedge b$ we mean the conjunction of the information represented by the intersection of the corresponding regions



and a subset U of S as the disjunction of the information in its elements, represented by the union of the regions of its elements. Then the covering can be understood by a picture: $a \lhd U$

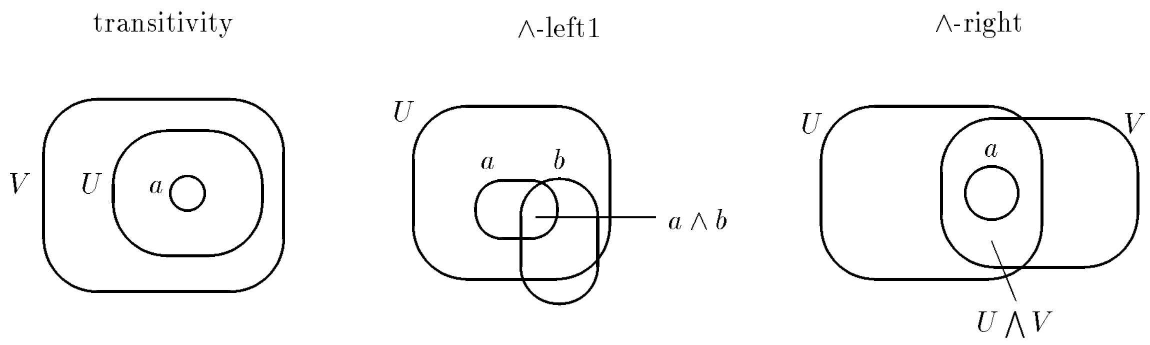
iff the region of a is covered by the region of U.



$a \lhd U$

where $U = \{b_1, b_2, b_3\}$

Transitivity, $\wedge$-left and $\wedge$-right can now be understood by the pictures



Thinking of the neighbourhoods in terms of information we can understand the information in a positive neighbourhood as meaningful or not contradictory. Monotonicity then says that if a is positive and a is covered by U, then U must contain something meaningful. Positivity says exactly that only positive elements contribute to the covering since positivity is equivalent to

$$\dfrac{a \lhd U}{a \lhd U^+}\,\text{openness} \qquad \text{where } U^+ \equiv \{b \in U : pos(b)\}$$

Proof: We first assume positivity and show openness:

$$\dfrac{a \lhd U \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[b \in U] \quad [pos(b)]}{b \in U^+}\,\text{def of } U^+}{b \lhd U^+}\,\text{reflexivity}}{pos(b) \to b \lhd U^+}\,\to\text{-intro}}{b \lhd U^+}\,\text{positivity}}{(\forall b \in U)(b \lhd U^+)}\,\forall\text{-intro}}{U \lhd U^+}\,\text{def}}{a \lhd U^+}\,\text{transitivity}$$

Then by assuming openness, positivity is proved by

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a \in \{a\}}}{a \lhd \{a\}}\,\text{reflexivity}}{a \lhd \{a\}^+}\,\text{openness} \qquad \dfrac{\dfrac{\dfrac{pos(a) \to a \lhd U \quad \dfrac{[b \in \{a\}^+]}{b = a \ \&\ pos(b)}\,\text{def}}{b \lhd U}\,\text{subst,}\to\text{-elim}}{(\forall b \in \{a\}^+)(b \lhd U)}\,\forall\text{-intro}}{\{a\}^+ \lhd U}\,\text{def}}{a \lhd U}\,\text{transitivity}$$

For the moment, regard the elements in S as being neighbourhoods of concrete points; $x\epsilon a$ will be used here to mean that a is a neighbourhood of the point x. Then 1 corresponds to the whole space, $\wedge$ corresponds to intersection, $a \lhd U$ means "the set of points forming a is included in the union of U", and $pos(a)$ means that a is inhabited. For this special case, we can actually prove monotonicity and positivity. For monotonicity: $pos(a)$ implies that there is a point, say x, in a and since $a \lhd U$ there exists a b in U such that $x\epsilon b$, that is $(\exists b \in U)\ pos(b)$. For positivity:

$$\dfrac{\dfrac{\dfrac{[x\epsilon a] \qquad \dfrac{\dfrac{\dfrac{[x\epsilon a]}{pos(a)}\,\text{pos-intro} \qquad pos(a) \to a \lhd U}{a \lhd U}\,\text{$\to$-elim}}{a \subseteq \bigcup U}\,\text{def}}{x\epsilon \bigcup U}\,\text{$\subseteq$-elim}}{a \subseteq \bigcup U}\,\text{$\subseteq$-intro}}{a \lhd U}\,\text{def}$$

2.3 Subsets as propositional functions

As mentioned before, we use propositional functions over the base set S as subsets of S. If U is a propositional function over S and a an element in S, then a is considered to be an element in U iff $U(a)$ holds. We extend this to explain when a subset (propositional function) is a subset of another subset of S. Let U and V be propositional functions over S, then U is a subset of V iff for all a in S, $U(a)$ implies $V(a)$. This can be defined by an introduction rule:

```
subset : (S:Set;U:(S)Set;V:(S)Set)Set

subsetintro : (S:Set;
               U:(S)Set;
               V:(S)Set;
               (a:S;U(a))V(a))
                 subset(S,U,V)
```

U and V are considered equal (as sets) iff they are subsets of each other:

```
eqsubset = [S,U,V]Product(subset(S,U,V),subset(S,V,U)) :
           (S:Set;U:(S)Set;V:(S)Set)Set
```

where `Product` is conjunction.

2.4 Using a context to formalize pointfree topology

We will now represent a formal topology by a list of assumptions (type declarations), in which we assume sets and functions ranging over these sets as well as express the axioms that describe the properties of the formal topology. Lists of type declarations are formalized as contexts, constructions which are governed by the following rules

$$[\,] : Context \qquad\qquad \dfrac{\Gamma : Context \quad \alpha : type\ [\Gamma]}{[\Gamma; x : \alpha] : Context}$$

where x does not occur free in Γ and $[\Gamma; x : \alpha]$ is the extension of Γ with the clause $x : \alpha$.

In the implementation `one` will be used for 1, `meet` and `MEET` for $\wedge$ and $\bigwedge$, respectively, `cov` and `COV` for $\triangleleft$ and $\triangleleft$, respectively.

First, `MEET`, `COV` and `POS` must be defined since they will be used inside the context defining the topology. They depend on `S`, `meet`, `cov` and `pos`, so `S`, `meet`, `cov` and `pos` occur as parameters in `MEET`, `COV` and `POS`. By this way `MEET`, `COV` and `POS` can be used to different contexts defining formal topologies. But standing for themselves, without such a context, they have of course not the intended meaning. `MEET`, `COV` and `POS` could be explicitly defined, using quantifiers, but introduction rules makes the proofs easier:

```
MEET : (S:Set;meet:(S;S)S;U:(S)Set;V:(S)Set;S)Set

MEETintro : (S:Set;meet:(S;S)S;U:(S)Set;V:(S)Set;a:S;b:S;U(a);V(b))
            MEET(S,meet,U,V,meet(a,b))

COV : (S:Set;cov:(S;(S)Set)Set;U:(S)Set;V:(S)Set)Set

COVintro : (S:Set;cov:(S;(S)Set)Set;U:(S)Set;V:(S)Set;(a:S;U(a))cov(a,V))
          COV(S,cov,U,V)

POS : (S:Set;pos:(S)Set;U:(S)Set)Set

POSintro : (S:Set;pos:(S)Set;U:(S)Set;b:S;U(b);pos(b))POS(S,pos,U)
```

Our context also makes use of singleton sets, which are explicitly defined using propositional equality:

```
Sing = Id : (S:Set;S;S)Set
```

Finally, the formal topology `TOP` is defined as a context which contains the following assumptions: `S` is a set with a particular element `1` and a binary operator `meet`, `cov` is a relation between elements and subsets of `S` and `pos` is a predicate on the elements of `S`, followed by the list of properties (corresponding to the rules in the definition of formal topology in section 2.1) that `S`, `1`, `meet`, `cov` and `pos` must have.

```
TOP is [S:Set; one:S; meet:(S;S)S; cov:(S;(S)Set)Set; pos:(S)Set;
        covmeet1:(a:S)cov(a,Sing(S,one));
        covrefl:(a:S;U:(S)Set;U(a))cov(a,U);
        covtrans:(a:S;
                  U:(S)Set;
                  V:(S)Set;
                  cov(a,U);
                  f:COV(S,cov,U,V))cov(a,V);
        covmeetl1:(a:S;b:S;U:(S)Set;cov(a,U))cov(meet(a,b),U);
        covmeetl2:(a:S;b:S;U:(S)Set;cov(b,U))cov(meet(a,b),U);
        covmeetr:(a:S;
                  U:(S)Set;
                  V:(S)Set;
                  cov(a,U);
                  cov(a,V))cov(a,MEET(S,meet,U,V));
        mono:(a:S;U:(S)Set;pos(a);cov(a,U))POS(S,pos,U);
        posi:(a:S;U:(S)Set;(pos(a))cov(a,U))cov(a,U)]
```

However, using contexts to represent algebraic structures have some drawbacks. For instance, the definition above gives us no template for making new topologies; a proof or definition that involve several algebraic structures require as many contexts. That means that reasoning using many algebraic structures is tedious. In Betarte [1] there is a more detailed discussion about this.

2.4.1 Concrete topology as substitution

We also want to express that some structure is an instance of the definition of formal topology. For that we use the notion of substitution, that is an assignment of objects of appropriate types to the variables in a context. Substitutions are introduced by the following rules

$$\{\,\} : [\,]\ [\Gamma] \qquad \frac{\gamma : \Delta\ [\Gamma] \quad \alpha : \mathit{type}\ [\Delta] \quad a : \alpha\gamma\ [\Gamma]}{\{\gamma; x := a\} : [\Delta; x : \alpha]\ [\Gamma]}$$

where $\{\,\}$ is the empty substitution and $\{\gamma; x := a\}$ is the extension of the substitution γ with the assignment $x := a$. This will be used in the example below. In Tasistro [15], substitutions are explained in more detail.

2.4.2 Example: Neighbourhoods of the natural numbers

As an example, given by Sambin [12], of a concrete pointfree topology we take the set SN of neighbourhoods of the natural numbers given by the rules

$$\frac{}{\overline{N} \in SN}$$

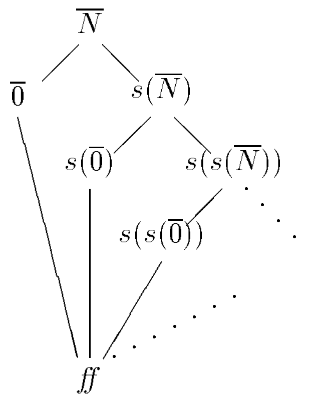
$$\frac{}{\overline{0} \in SN}$$

$$\frac{a \in SN}{s(a) \in SN}$$

$$\frac{}{\mathit{ff} \in SN}$$

and if a and b are two neighbourhoods of a number then, their intersection, $a \wedge_{nat} b$ is a neighbourhood of the same number. Furthermore, a neighbourhood is positive if it is a neighbourhood of a number.

The intended meaning is that $s^n(\overline{N})$, where $n \in N$, is a neighbourhood of all numbers in $\{n, n+1, n+2, ...\}$, $s^n(\overline{0})$ is a neighbourhood only of $s^n(0)$, and no number has ff as neighbourhood (ff is needed to make sure that given two neighbourhoods a and b, $a \wedge_{nat} b$ is a neighbourhood). The figure illustrates the structure that the neighbourhoods form:



The figure is not complete, there are also an infinite number of empty neighbourhoods of the form $s(...s(ff)...)$, which are not identical to ff but are equal to ff in the sense that they are all non positive and therefore also covered by each other's singleton sets.

Formalized in type theory, SN is a set with four constructors:

```
SN : Set
onenat : SN
zero : SN
s : (SN)SN
ff : SN
```

where `onenat` and `zero` correspond to $\overline{N}$ and $\overline{0}$, respectively. $\wedge_{nat}$ (`meetnat`) can be implicitly defined, using pattern matching:

```
meetnat : (a:SN;b:SN)SN
        meetnat(onenat,b) = b
        meetnat(zero,onenat) = zero
        meetnat(zero,zero) = zero
        meetnat(zero,s(h)) = ff
        meetnat(zero,ff) = ff
        meetnat(s(h),onenat) = s(h)
        meetnat(s(h),zero) = ff
        meetnat(s(h),s(h1)) = s(meetnat(h,h1))
        meetnat(s(h),ff) = ff
        meetnat(ff,b) = ff
```

We define a neighbourhood to be positive if it is a neighbourhood of a number, thus ff, $s(ff)$, $s(s(ff))$, ... are the only non-positive neighbourhoods:

```
posnat : (a:SN)Set
        posnat(onenat) = N1
        posnat(zero) = N1
        posnat(s(h)) = posnat(h)
        posnat(ff) = Empty
```

where `N1` is the set containing `tt` as only element, that is, a true proposition and `Empty` is the empty set, that is, a false proposition.

Before defining the covering relation we define a partial order $\leq_{nat}$ on the neighbourhoods by

$$a \leq_{nat} b \quad \text{iff} \quad a \wedge_{nat} b = a$$

This is the same ordering as in a semilattice (and in the figure above), which the neighbourhoods in fact form even though we have not proved it yet. In type theory $\leq_{nat}$ is explicitly defined using propositional equality:

```
leqnat = [a,b]Id(SN,meetnat(a,b),a) : (a:SN;b:SN)Set
```

Now we can define the covering relation, for arbitrary $a \in S$ and $U \subseteq S$, by

$$a \lhd U \quad \text{iff} \quad a \textit{ is not positive} \quad \text{or} \quad (\exists b \in U)(a \leq_{nat} b)$$

But instead the following definition by introduction rules will be used

```
covnat : (a:SN;U:(SN)Set)Set
covnati1 : (a:SN;U:(SN)Set;(posnat(a))Empty)covnat(a,U)
covnati2 : (a:SN;U:(SN)Set;b:SN;U(b);leqnat(a,b))covnat(a,U)
```

It is easy to see that the two definitions of covering above ($\lhd$ and `covnat`) are equivalent. The reason not to define `covnat` explicitly, using existential quantification, is that the definition by introduction rules makes the proofs easier and shorter.

In order to show that `SN`, `onenat`, `meetnat`, `covnat` and `posnat` is a formal topology one must prove that all the properties of formal topology (the properties listed in the definition of `TOP`) are satisfied. Consult appendix C for more details.

The proof that the neighbourhoods of the natural numbers is a formal topology is then completed by the substitution `TOPNAT`:

```
TOPNAT is {S:=SN; one:=onenat; meet:=meetnat; cov:=covnat; pos:=posnat;
           covmeet1:=covmeetnat1; covrefl:=covreflnat;
           covtrans:=covtransnat; covmeetl1:=covmeetnatl1;
           covmeetl2:=covmeetnatl2; covmeetr:=covmeetnatr;
           mono:=mononat; posi:=posinat} : TOP     []
```

2.5 Properties of a formal topology

In this section we will concentrate on definitions and types, not on the proofs. The proof terms of the types are too long for a readable presentation, they can however be obtained by ftp (see the introduction). For a description of the proofs see Sambin [12]. The definitions and results of this section are not used in the rest of the paper.

2.5.1 Frames and complete Heyting algebras

Here we show that a formal topology defines a frame in such a way that equivalence classes of subsets (the equality will soon be defined) are the opens, `COV` corresponds to the partial order and `MEET` corresponds to the meet operation.

First we define the equality relation between subsets such that two subsets are equal iff they cover each other:

```
EQS = [U,V]Product(COV(S,cov,U,V),COV(S,cov,V,U)) :
       (U:(S)Set;V:(S)Set)Set     TOP
```

Note here that we are doing all this in the context `TOP`. That `EQS` is an equivalence relation is easily proved (see appendix E: `EQSsymm`, `EQSrefl`, `EQStrans`).

The opens (equivalence classes of subsets) are difficult to define in ALF and so are ordering, meet- and join-operations for opens, instead we will rely on the fact that the ordering respects `EQS` and that `EQS` respects meet and join, which are defined on subsets. Of course that has to be proved, the types of the proof is in appendix E: `COVrespEQS`, `EQSrespMEET`, `EQSrespJOIN`.

For the ordering `COV` is used, which is a partial order on the family of subsets of `S` (appendix E: `COVtrans`, `COVrefl`, antisymmetry follows directly from the definition of the equality `EQS`).

For the meet operation we use `MEET`, which gives the infimum (appendix E: `MEETisinfl1`, `MEETisinfl2`, `MEETisinfr`).

Join is defined as a union:

```
JOIN = [T,I,U]union(S,T,I,U) : (T:Set;I:(T)Set;U:(T;S)Set;S)Set     TOP
```

We postpone the definition of `union` to section 3.4. `JOIN` gives the supremum (appendix E: `JOINissup1`, `JOINissup2`).

Finally the infinite distributivity

```
(T:Set;I:(T)Set;V:(S)Set;U:(T;S)Set)
  EQS(MEET(S,meet,V,JOIN(T,I,U)),
      JOIN(T,I,[i]MEET(S,meet,V,U(i))))     TOP
```

holds (appendix E: `infdistr`).

This far we have proved that a formal topology defines a frame. Implication can then be defined in the frame so it becomes a *complete Heyting algebra*.

A *complete Heyting algebra* is a complete lattice A where, for every $a, b \in A$, there is an element $a \to b$ satisfying

$c \leq a \to b$ iff $c \wedge a \leq b$.

In a frame $\to$ is defined by

$a \to b \equiv \bigvee\{c : c \wedge a \leq b\}$.

For a proof that this definition of implication gives a complete Heyting algebra see for instance [16].

The definition of implication translated to our case becomes

```
cHaimply = [U,V,a]COV(S,cov,MEET(S,meet,U,Sing(S,a)),V) :
              (U:(S)Set;V:(S)Set;a:S)Set     TOP
```

`cHaimply` respects `EQS` and satisfies the implication property (see appendix E: `cHaimplyrespEQS`, `cHaimplyprop1`, `cHaimplyprop2`). This completes the proof that a formal topology defines a complete Heyting algebra.

2.5.2 Closure operator

In the previous subsection it was shown that the equivalence classes of subsets form a frame. Now we will define a *closure operator*, that is an operator that given a subset U returns its *downward closure*. The downward closure of a subset U is the subset of all neighbourhoods which are covered by U.

We will show that each equivalence class contains a closed set and that the closed sets form a frame which is isomorphic to the frame formed by the equivalence classes in such way that each equivalence class is represented by its closed set.

A closure operator, Cl, is an operator acting on subsets and satisfying the following properties

$$U \subseteq Cl(U)$$
$$U \subseteq V \to Cl(U) \subseteq Cl(V)$$
$$Cl(Cl(U)) = Cl(U)$$

Here `Cl` is explicitly defined by

```
Cl = [U,a]cov(a,U) : (U:(S)Set;a:S)Set      TOP
```

`Cl` satisfy the closure operator properties (appendix F: `Clprop1`,2,3).

We then say that a subset is closed or saturated if it is equal, as a subset, to its closure:

```
sat = [U]eqsubset(S,U,Cl(U)) : (U:(S)Set)Set      TOP
```

Since

```
(U:(S)Set)EQS(U,Cl(U))      TOP
```

holds, any equivalence class contains a closed subset. Given two subsets in the same equivalence class, their closures are equal

```
(U:(S)Set;V:(S)Set;EQS(U,V))eqsubset(S,Cl(U),Cl(V))      TOP,
```

so any equivalence class contains exactly one closed subset. Thus the closed subsets form a frame which is isomorphic to the frame formed by the equivalence classes.

```
(U:(S)Set;V:(S)Set;COV(S,cov,U,V))subset(S,Cl(U),Cl(V))      TOP
```

and

```
(U:(S)Set;V:(S)Set;subset(S,Cl(U),Cl(V)))COV(S,cov,U,V)      TOP
```

hold, so the order in this frame is the subset order.

`cHaimply(U,V)` is closed for any two subsets `U` and `V`, and `Cl` preserves implication, so the closed sets form a cHa which is isomorphic to the one formed by the equivalence classes (appendix F: `satcHaimply`, `ClprescHaimply`).

In appendix F meet- and join-operations are also defined. In appendix F there are also proofs of that `Cl` is a cHa isomorphism.

2.6 Points

A formal point (Sambin [13]) of a formal topology $\langle S, \wedge, 1, \lhd, pos \rangle$ is a subset p of S which, for arbitrary $a, b \in S$, satisfies

1. $\dfrac{}{1 \in p}$

2. $\dfrac{a \in p \quad b \in p}{a \wedge b \in p}$

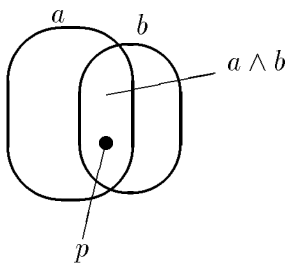
3. $\dfrac{a \in p \quad a \lhd U}{(\exists b \in U)(b \in p)}$

4. $\dfrac{a \in p}{pos(a)}$

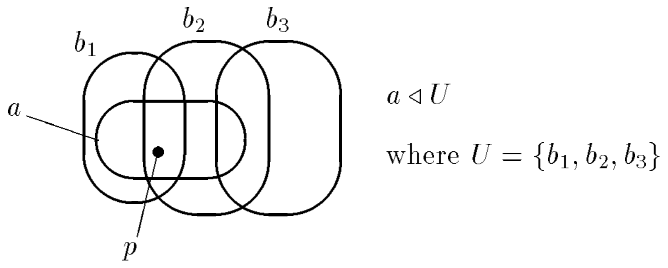
Even though a point is a subset, the intuition of a subset as an open and a subset as a point are not the same. A subset (recall section 2.2) we regard as union of the regions of its elements, while we can understand a point as something in the intersection of all neighbourhoods in it. So an informal understanding of $a \in p$ (where a is a neighbourhood and p a point) might be "p is a point in a". Then we can understand the definition of points in the following way.

1. Any point p is in the space (since 1 corresponds to the whole space).

2. If p is in both a and b, then p is in the intersection of a and b.



3. If p is in a and a is covered by U, then U must contain a neighbourhood containing p.



4. If p is in a then a is meaningful.

To avoid the existential quantification, in rule 3 of the definition of formal point, we make the following definition

```
P : (p:(S)Set;U:(S)Set)Set      TOP

Pintro : (p:(S)Set;U:(S)Set;b:S;U(b);p(b))P(p,U)      TOP
```

Informally: $P(p, U)$ holds iff $(\exists b \in U)(b \in p)$.

From rule (3) one can see that a definition by introduction rules of points impossible:

```
point : (p:(S)Set)Set     TOP

pointintro : (p:(S)Set;
              p(one);
              (a:S;b:S;p(a);p(b))p(meet(a,b));
              (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
              (a:S;p(a))pos(a))
                point(p)     TOP
```

For instance it does not follow the general scheme (given in [5]) of an inductive definition: `U` is of function type and is not a parameter to the definition. So a neat definition of formal points in type theory seems to be impossible. Instead we can do the following: in order to prove that a subset `p` is a point we prove

```
p(one) ,

(a,b:S;p(a);p(b))p(meet(a,b)) ,

(a:S;U:(S)Set;p(a);cov(a,U))P(p,U)
```

and

```
(a:S;p(a))pos(a).
```

And in order to prove that, given a point `p`, some property `C(p)` holds, we assume all properties a point must have:

```
(p:(S)Set;
 p(one);
 (a:S;b:S;p(a);p(b))p(meet(a,b));
 (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
 (a:S;p(a))pos(a))
   C(p)     TOP
```

The above definition of points corresponds exactly to the definition of points as completely prime filters. For details see appendix G.

3 The points of a Scott topology form a Scott domain

In this section we will show that the formal points of a *Scott topology* form a *Scott domain.*

3.1 Scott domain

By a Scott domain we mean an algebraic cpo in which every family of elements which is bounded above has a least upper bound. Observe that we use the word family and not subset: in general the points do not form a proper set in the type theoretic sense. In the following definitions, which are adopted from Sambin [13], there is a distinction between *sets*, *collections* and *families*. Sets are inductively defined and families are subcollections indexed by sets or subsets (propositional functions).

Let $\mathcal{D} = \langle D, \sqsubseteq \rangle$ be a partially ordered collection. A family $(x_i)_{i \in I}$ of elements in D is *bounded* whenever there exist an element $x \in D$ such that $(\forall i \in I)(x_i \sqsubseteq x)$ and *directed* if I is inhabited and $(\forall i, j \in I)(\exists k \in I)(x_i \sqsubseteq x_k \ \& \ x_j \sqsubseteq x_k)$. $\mathcal{D}$ is called a *complete partial order* (*cpo*) if D has a minimum element $\perp$ and every directed family has a supremum. The supremum of a directed family $(x_i)_{i \in I}$ will be denoted $\bigsqcup_{i \in I} x_i$.

An element a of a cpo $\mathcal{D}$ is called *compact* if, for any directed family $(x_i)_{i \in I}$ of elements in D, $a \sqsubseteq \bigsqcup_{i \in I} x_i$ implies that $(\exists k \in I)(a \sqsubseteq x_k)$. We will write $K(D)$ for the collection of compact elements of D.

A cpo $\mathcal{D}$ is called *algebraic* if, for every $x \in D$, the collection $\{a \in K(D) : a \sqsubseteq x\}$ of compact lower bounds of x is a directed family of elements $(a_i)_{i \in I}$, for a suitable index set I, such that $x = \bigsqcup_{i \in I} a_i$. This definition is stronger than the traditional, normally it is only required that $x = \bigsqcup \{a \in K(D) : a \sqsubseteq x\}$ since $\{a \in K(D) : a \sqsubseteq x\}$ is directed. But here we also require that the compact elements of a domain must form a family.

From Sambin [13] it follows that any algebraic cpo such that any bounded pair of compact elements has a supremum is a Scott domain. This is the property that we will show that the points satisfy.

3.2 Scott topology

In the following definition we mean by set, set in classical set theory. Let $\langle X, \sqsubseteq \rangle$ be a poset of points. The *Scott topology* on $\langle X, \sqsubseteq \rangle$ consists of all sets $U \subseteq X$ that satisfy

- U is upward closed, that is if $x \in U$ and $x \sqsubseteq y$ then $y \in U$.
- U is inaccessible by directed joins, that is if $V \subseteq X$ is directed and $\bigvee V \in U$ then $(\exists x \in V)(x \in U)$.

Now let $\langle X, \sqsubseteq \rangle$ be a Scott domain and consider its Scott topology. It can be shown (Sambin [13]) that the subsets $O_U = \{x \in X : (\forall a \in U)(a \sqsubseteq x)\}$, for $U \subseteq_f K(X)$ ($\subseteq_f$ means finite subset), form a base for this topology. Moreover if O_U is inhabited and $O_U \subseteq \bigcup_{i \in I} O_{U_i}$ then $(\exists i \in I)(O_U \subseteq O_{U_i})$.

Proof: Assume O_U is inhabited and $O_U \subseteq \bigcup_{i \in I} O_{U_i}$. From O_U inhabited it follows that U is bounded above and since X is a Scott domain U has a supremum $\bigsqcup U$, for which $(\forall a \in U)(a \sqsubseteq \bigsqcup U)$ holds, that is $\bigsqcup U \in O_U$. Now

$$\begin{aligned} O_U \subseteq \bigcup_{i \in I} O_{U_i} \quad &\Rightarrow \quad \bigsqcup U \in \bigcup_{i \in I} O_{U_i} \\ &\Leftrightarrow \quad (\exists i \in I)(\bigsqcup U \in O_{U_i}) \\ &\Leftrightarrow \quad (\exists i \in I)(\forall a \in U_i)(a \sqsubseteq \bigsqcup U). \end{aligned}$$

Then take an arbitrary $x \in O_U$. Since $\langle X, \sqsubseteq \rangle$ is algebraic, x is equal to the supremum of its compact lower bounds, hence $\bigsqcup U \sqsubseteq x$. By transitivity of $\sqsubseteq$, $(\forall a \in U_i)(a \sqsubseteq x)$ for some $i \in I$, that is $x \in O_{U_i}$ for some $i \in I$. So $(\exists i \in I)(O_U \subseteq O_{U_i})$.

This property of Scott topologies is taken as definition in the pointfree approach, thinking of the base $\{O_U : U \subseteq_f K(X)\}$ as a formal base. A formal topology is called *Scott* if it satisfies

$$\frac{a \lhd U \quad pos(a)}{(\exists b \in U)(a \lhd \{b\})}\ \text{scott}$$

By the definition

```
  SCOTTOP is TOP + [scott:(a:S;
                           U:(S)Set;
                           cov(a,U);
                           pos(a))
                              Exists(S,[b]Product(U(b),cov(a,Sing(S,b))))]
```

SCOTTOP is the context TOP extended with the scott property.

3.2.1 Example: Neighbourhoods of the natural numbers

Recall the example in section 2.4.2. It is easy to see that the Scott property is also satisfied, so SN, onenat, meetnat, covnat and posnat actually form a Scott topology. For a full proof in ALF see appendix I.

3.3 Points of a Scott topology

If the Scott property holds then the covering relation $\lhd$ can be replaced by the simpler relation $\leq$ defined by

$$a \leq b \equiv a \lhd \{b\}$$

A formal point is the same as a subset p satisfying

1. $\dfrac{}{1 \in p}$
2. $\dfrac{a \in p \quad b \in p}{a \wedge b \in p}$
3. $\dfrac{a \in p \quad a \leq b}{b \in p}$
4. $\dfrac{a \in p}{pos(a)}$

If we first define the new order

```
  leq = [a,b]cov(a,Sing(S,b)) : (a:S;b:S)Set     TOP
```

then the points can be defined (there is no longer a quantification over subsets):

```
  scpoint : (p:(S)Set)Set     SCOTTOP

  scpintro : (p:(S)Set;
              p(one);
              (a:S;b:S;p(a);p(b))p(meet(a,b));
              (a:S;b:S;p(a);leq(a,b))p(b);
              (a:S;p(a))pos(a))
                scpoint(p)     SCOTTOP
```

It is easy to prove that if the Scott property holds then a subset p is a point iff scpoint(p) holds. For details see appendix J (point2scpoint, scpoint2point).

3.4 Definition of union of subsets, directed families and compactness

Before we present the proof, that the points of a Scott topology form a Scott domain, some more definitions are needed. In the proofs we will look at families of points where the index set itself is a point (propositional function), so in the definitions the index set must be a general subset. Formally, the family $\{p_i\}_{i \in I}$ consists of three parts:

```
T:Set
I:(T)Set
p:(T;S)Set
```

The intended meaning is: if `i:T` then `p(i)` is a member of the family iff `I(i)` holds.

Union of subsets:

```
union : (S:Set;T:Set;I:(T)Set;U:(T;S)Set;S)Set

unionintro : (S:Set;
              T:Set;
              I:(T)Set;
              U:(T;S)Set;
              i:T;
              I(i);
              a:S;
              U(i,a))
                union(S,T,I,U,a)
```

The order in the domain is the subset order, so in the definition of directed families and compactness we use `subset`.

Directed families:

```
directed : (S:Set;T:Set;I:(T)Set;p:(T;S)Set)Set

directedintro : (S:Set;
                 T:Set;
                 I:(T)Set;
                 p:(T;S)Set;
                 i0:T;
                 I(i0);
                 (i:T;j:T;I(i);I(j))
                   Exists(T,[k]Product(I(k),
                                       Product(subset(S,p(i),p(k)),
                                               subset(S,p(j),p(k)))))
                     directed(S,T,I,p)
```

As was the case with the points (section 2.6), a definition by introduction rules of compactness is impossible. It would contain quantifications over function types:

```
compact : ((S)Set)Set

compactintro : (q:(S)Set;
                (T:Set;
                 I:(T)Set;
                 p:(T;S)Set;
                 directed(S,T,I,p);
                 subset(S,q,union(S,T,I,p)))
                   Exists(T,[i]Product(I(i),subset(S,q,p(i)))))
                     compact(q)
```

In order to prove that a point q in a cpo is compact, one has to show that the function type

```
(T:Set;
 I:(T)Set;
 p:(T;S)Set;
 directed(S,T,I,p);
 subset(S,q,union(S,T,I,p)))
   Exists(T,[i]Product(I(i),subset(S,q,p(i))))
```

is inhabited. And in order to prove that, given a compact point q, some property C(q) holds then all the properties of a compact point has to be assumed:

```
(q:(S)Set;
 scpoint(q);
 (T:Set;
  I:(T)Set;
  p:(T;S)Set;
  directed(S,T,I,p);
  subset(S,q,union(S,T,I,p)))Exists(S,[i]Product(I(i),subset(S,q,p(i)))))
    C(q)
```

3.5 The points of a Scott topology form a Scott domain

Unless otherwise stated, the types for the propositions in this section can be found in appendix L. The formal proofs can be obtained by ftp.

3.5.1 The points form an algebraic cpo

In order to show that the points of a Scott topology form a cpo, which we denote by $\mathcal{P}t(S)$ (where S is the formal base), we need the extra property that 1 is positive. Points are subsets of positive neighbourhoods and 1 is contained in all points, so without the knowledge that 1 is positive we cannot show that there are any points at all, particularly no bottom element, and consequently no cpo.

The order is the subset order (subset) which, of course, is reflexive (subsetrefl in appendix A), transitive (subsettrans in appendix A) and antisymmetric (by definition of the equality, eqsubset).

Next we need a bottom element, a point which is a subset of all other points. Given a positive element $a \in S$, its upper closure $\uparrow a = \{c \in S : a \leq c\}$ can easily be shown to be a point. In our type theoretic notation, the upper closure of a neighbourhood `a` is `leq(a)`. `genscp` is a proof that given an arbitrary positive neighbourhood `a`, `leq(a)` is a point; in particular if `pos(one)` holds then `leq(one)` is a point. Intuitively, since all points contains 1 and all points are upper closed, the upper closure of 1 is a least element; `leqonemin` is a formal proof of that.

The union of a family of subsets is, of course, an upper bound of the family (`unionsup1`), it is less than or equal to all upper bounds (`unionsup2`) and the union of a directed family of points is a point itself (`unionpoint`). So given a directed family of points its supremum is formed by taking the union of all points in the family.

That $\mathcal{P}t(S)$ is algebraic is proved as follows. Given two arbitrary points p and q, we have

$$(\uparrow a)_{a\in p} \textit{ is directed} \tag{1}$$

and

$$q \subseteq p \;\&\; q \textit{ compact} \Leftrightarrow (\exists a \in p)(q = \uparrow a) \tag{2}$$

which implies that the family of compact lower bounds to p is $(\uparrow a)_{a\in p}$. Finally the union of $(\uparrow a)_{a\in p}$ (which is the supremum) is equal to p:

$$\bigcup_{a\in p} \uparrow a = p \tag{3}$$

Proof of 1: $1 \in p$ so p is inhabited. If $a, b \in p$ then $a \wedge b \in p$ and $\uparrow a, \uparrow b \subseteq \uparrow(a \wedge b)$.

Proof of 2 $\Rightarrow$: Assume that $q \subseteq p$ and q is compact. The family $(\uparrow a)_{a\in q}$ is directed (follows from 1). Clearly $q \subseteq \bigcup_{a\in q} \uparrow a$, so by the definition of compactness $(\exists a \in q)(q \subseteq \uparrow a)$ follows. But if $a \in q$ then $\uparrow a \subseteq q$, so we have $(\exists a \in q)(q = \uparrow a)$ and from the assumption $q \subseteq p$ we get $(\exists a \in p)(q = \uparrow a)$.

$\Leftarrow$: Assume that $(\exists a \in p)(q = \uparrow a)$. Clearly $q \subseteq p$ holds. By existential elimination $q = \uparrow a$, for some $a \in p$. Let $(r_i)_{i\in I}$ be a directed family such that $q \subseteq \bigcup_{i\in I} r_i$. By substitution we have $\uparrow a \subseteq \bigcup_{i\in I} r_i$. Now

$$\begin{aligned} \uparrow a \subseteq \bigcup_{i\in I} r_i \quad &\Leftrightarrow \quad a \in \bigcup_{i\in I} r_i \\ &\Leftrightarrow \quad (\exists i \in I)(a \in r_i) \\ &\Leftrightarrow \quad (\exists i \in I)(\uparrow a \subseteq r_i). \end{aligned}$$

And by substituting back $(\exists i \in I)(q \subseteq r_i)$ follows. So q is compact.

Proof of 3: It is easy to see that $p \subseteq \bigcup_{a\in p} \uparrow a$. Let $b \in \bigcup_{a\in p} \uparrow a$ then $(\exists a \in p)(b \in \uparrow a)$. But if $a \in p$ then $\uparrow a \subseteq p$ so $b \in p$. Thus $\bigcup_{a\in p} \uparrow a \subseteq p$.

Again, the proof terms are too long so we only present the types. 1 is proved by

```
genscpdir : (p:(S)Set;scpoint(p))directed(S,S,p,leq)     SCOTTOP
```

The implication from left to right in 2 follows from

```
scpcomplb1 : (q:(S)Set;
             scpoint(q);
             (T:Set;
              I:(T)Set;
              r:(T;S)Set;
              directed(S,T,I,r);
              subset(S,q,union(S,T,I,r)))
                Exists(T,[x]Product(I(x),subset(S,q,r(x))));
             p:(S)Set;
             subset(S,q,p))
               Exists(S,[a]Product(p(a),eqsubset(S,q,leq(a))))     SCOTTOP
```

The first conjunct in the implication from right to left follows from

```
scpcomplb2a : (p:(S)Set;
              q:(S)Set;
              scpoint(p);
              Exists(S,[a]Product(p(a),eqsubset(S,q,leq(a)))))
                subset(S,q,p)     SCOTTOP
```

and the second conjunct from

```
scpcomplb2b : (p:(S)Set;
              q:(S)Set;
              Exists(S,[a]Product(p(a),eqsubset(S,q,leq(a))));
              T:Set;
              I:(T)Set;
              r:(T;S)Set;
              (i:T;I(i))scpoint(r(i));
              subset(S,q,union(S,T,I,r)))
                Exists(T,[x]Product(I(x),subset(S,q,r(x))))     SCOTTOP
```

3 is proved by

```
supcompscp : (p:(S)Set;scpoint(p))eqsubset(S,p,union(S,S,p,leq))     SCOTTOP
```

3.5.2 Every bounded pair of compact points has a supremum

In order to prove that every bounded pair of compact points has a supremum we first notice that

$$\textit{if a point } p \textit{ is compact then } (\exists a \in p)(p = \uparrow a) \qquad (4)$$

The proof of this is similar to the proof of 2. (In fact the converse also holds). Now take two arbitrary compact points p_1 and p_2, we then know that there exists positive a and b such that $p_1 = \uparrow a$ and $p_2 = \uparrow b$. If p_1 and p_2 are bounded, by say the point r, then $a, b \in r$ and since r is a point $a \wedge b \in r$. Again, since r is a point, $a \wedge b$ is positive so $\uparrow(a \wedge b)$ is a point. $\uparrow a, \uparrow b \subseteq \uparrow(a \wedge b)$ and if $\uparrow a, \uparrow b \subseteq q$ then $\uparrow(a \wedge b) \subseteq q$. Hence the supremum of p_1 and p_2 is $\uparrow(a \wedge b)$, provided they are bounded.

4 follows from

```
  gencompscp : (p:(S)Set;
                scpoint(p);
                (T:Set;
                 I:(T)Set;
                 p2:(T;S)Set;
                 directed(S,T,I,p2);
                 subset(S,p,union(S,T,I,p2)))
                   Exists(T,[h]Product(I(h),subset(S,p,p2(h)))))
                 Exists(S,[h]Product(p(h),eqsubset(S,p,leq(h))))
```

Now the fact that every two compact points, which are bounded above, have a supremum is proved by

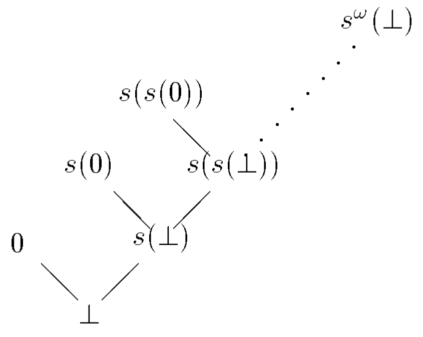
```
  psuptoleqp : (p1:(S)Set;
                p2:(S)Set;
                scpoint(p1);
                scpoint(p2);
                (T:Set;
                 I:(T)Set;
                 q:(T;S)Set;
                 directed(S,T,I,q);
                 subset(S,p1,union(S,T,I,q)))
                   Exists(T,[h]Product(I(h),subset(S,p1,q(h))));
                (T:Set;
                 I:(T)Set;
                 q:(T;S)Set;
                 directed(S,T,I,q);
                 subset(S,p2,union(S,T,I,q)))
                   Exists(T,[h]Product(I(h),subset(S,p2,q(h))));
                r:(S)Set;
                scpoint(r);
                subset(S,p1,r);
                subset(S,p2,r);
                q:(S)Set;
                scpoint(q))
                  Exists(S,[x]Product(Product(scpoint(leq(x)),
                                              Product(subset(S,p1,leq(x)),
                                                      subset(S,p2,leq(x)))),
                                      Imply(Product(subset(S,p1,q),
                                                    subset(S,p2,q)),
                                            subset(S,leq(x),q))))     SCOTTOP
```

The meaning of the type above is the following: if p_1, p_2 and q are compact points, such that p_1 and p_2 are bounded above, then

$$(\exists x \in S)(\uparrow x \textit{ is a point } \& \ p_1, p_2 \subseteq \uparrow x \ \& \ (p_1, p_2 \subseteq q \ \rightarrow \uparrow x \subseteq q)).$$

3.5.3 Example: Natural numbers

Recall again the example in section 2.4.2 and 3.2.1, on formal neighbourhoods of the natural numbers. From the definition of points in a Scott topology it is easy to see that the domain formed by the points in our case is



where

$\bot$	is	$\{\overline{N}\}$
0	is	$\{\overline{N}, \overline{0}\}$
$s(\bot)$	is	$\{\overline{N}, s(\overline{N})\}$
$s(0)$	is	$\{\overline{N}, s(\overline{N}), s(\overline{0})\}$
$s(s(\bot))$	is	$\{\overline{N}, s(\overline{N}), s(s(\overline{N}))\}$
.		.
.		.
$s^\omega(\bot)$	is	$\{\overline{N}, s(\overline{N}), s(s(\overline{N})), ...\}$

and all points except of $s^\omega(\bot)$ are finite.

4 Discussion

4.1 Subsets as propositional functions

By using propositional functions to represent subsets we can form subsets that cannot be constructed by using sets of type `Set` as subsets: if P is a predicate over the set S then we also have the subset $\{x \in S : P(x)\}$, in general there is no way to effectively produce the elements of this subset, and the same element can occur in several subsets.

Another possibility is to use $\sum$-sets: let $a \in \sum(S, U)$ iff there exist some $b : U(a)$ such that $\langle a, b\rangle : \sum(S, U)$. But to create $\sum(S, U)$ we need the predicate U and when we use an element a of a subset we first have to pick out a from the pair $\langle a, b\rangle$.

4.2 The consistency predicate

The rule of positivity,

$$\frac{pos(a) \to a \lhd U}{a \lhd U} \quad ,$$

has not been used in any formal proof. The rule of monotonicity,

$$\frac{pos(a) \quad a \lhd U}{(\exists b \in U)\ pos(b)}\ ,$$

has been used to show: if a is a positive neighbourhood in a Scott topology, then the upper closure of a, $\uparrow a$, is a point. This, in turn, is frequently used in the proof that the points of a Scott topology form a Scott domain, but the reason for that is that the points, by definition, consist of positive neighbourhoods.

By simply removing the consistency predicate and its rules, we can still show that "any formal topology defines a frame" and "the points of a Scott topology form a Scott domain". For Scott topologies, however, the Scott property,

$$\frac{a \lhd U \quad pos(a)}{(\exists b \in U)(a \lhd \{b\})}\ ,$$

must be changed; if a is the least element then $a \lhd U$ even if U is empty. A new condition for a Scott topology might be

$$a \lhd U \quad \leftrightarrow \quad (\forall b \in S)(a \lhd \{b\}) \ \bigvee\ (\exists b \in U)(a \lhd \{b\})$$

or, even better, remove the old condition and replace the covering $\lhd$ by $\leq$ (defined by $a \leq b \equiv a \lhd \{b\}$) in all the other rules. So one might ask whether the consistency predicate is needed. The category of Scott topologies (with consistency predicate) is equivalent to the category of Scott domains (for a proof see [13]), we cannot expect that this equivalence still holds if we remove some rules from the definition of formal topology.

4.3 Problems

One of the main problems that occured when formalizing pointfree topology and domain theory in ALF, was that we did not find any internal definition of points (section 2.6) and compactness (section 3.4). As a consequence many types have become long and hard to read; it is cumbersome to say that something is a point/compact element, both as assumption and as result of a proposition. Another consequence is that the proof terms, even for trivial lemmas, have become unreadable; they contain many variables and several of them are of function type.

Another problem is that some properties are difficult to express inside the theory. To show the statements "the equivalence classes of subsets form a frame" and "the points of a Scott topology form a Scott domain" we have proved a lot of properties, which together implies that one can understand, outside the theory, that the statements are correct.

5 Acknowledgements

I would like to thank my supervisor Jan Smith for all his support. I also would like to thank Thierry Coquand, Bengt Nordström, Randy Pollack, Mikael Rittri and Björn von Sydow for discussions and valuable comments on earlier versions of this thesis.

References

[1] Gustavo Betarte. A case study in machine-assisted proofs: The integers form an integral domain. Licentiate Thesis, Chalmers University of Technology and University of Göteborg, Sweden, November 1993.

[2] Thierry Coquand. Constructive topology and combinatorics. In *proceeding of the conference Constructivity in Computer Science, San Antonio, LNCS 613*, pages 28–32, 1992.

[3] Thierry Coquand. Pattern matching with dependent types. In *Proceeding from the logical framework workshop at Båstad*, June 1992.

[4] Thierry Coquand. An intuitionistic proof of tychonoff's theorem. *Journal of Symbolic Logic*, pages 28–32, Volume 57, 1992.

[5] Peter Dybjer. An inversion principle for Martin-Löf's type theory. In *Proceedings of the Workshop on Programming Logic, Båstad*, May 1989. Accepted (subject to revision) for publication in *Formal Aspects of Computing.*

[6] Michael Hedberg. *Type Theory and the External Logic of Programs.* PhD thesis, Department of Computing Science, Chalmers University of Technology, Göteborg, Sweden, May 1994.

[7] Peter T. Johnstone. *Stone spaces.* Cambridge University Press, 1982.

[8] John L. Kelley. *General Topology.* Van Nostrand, 1955.

[9] Lena Magnusson. The new Implementation of ALF. In *The informal proceeding from the logical framework workshop at Båstad, June 1992*, 1992.

[10] Per Martin-Löf. The Domain Interpretation of Type Theory, Lecture Notes. In Kent Karlsson and Kent Petersson, editors, *Workshop on Semantics of Programming Languages, Abstracts and Notes*, Chalmers University of Technology and University of Göteborg, August 1983. Programming Methodology Group.

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

[12] Giovanni Sambin. Intuitionistic Formal Spaces - A First Communication. In *The Proceedings of Conference on Logic and its Applications, Bulgaria.* Plenum Press, 1986.

[13] Giovanni Sambin, Silvio Valentini, and Paolo Virgili. Constructive domain theory as a branch of intuitionistic pointfree topology. Technical report, Dip.di Matematica Pura e Appl., Univ. di Padova, August 1992.

[14] Dana Scott. Lectures on a Mathematical Theory of Computation. Technical Monograph, Prg 19, Oxford University Computing Laboratory, May 1981.

[15] Alvaro Tasistro. Formulation of Martin-Löf's Theory of Types with Explicit Substitution. Licentiate Thesis, Chalmers University of Technology and University of Göteborg, Sweden, May 1993.

[16] Steven Vickers. *Topology Via Logic.* Cambridge University Press, 1989.

A Subsets as propositional functions

```
subset : (S:Set;U:(S)Set;V:(S)Set)Set     []    C

subsetintro : (S:Set;
               U:(S)Set;
               V:(S)Set;
               (a:S;U(a))V(a))
                 subset(S,U,V)     []    C


eqsubset = [S,U,V]Product(subset(S,U,V),subset(S,V,U)) :
           (S:Set;U:(S)Set;V:(S)Set)Set     []



subsetrefl : (S:Set;U:(S)Set)subset(S,U,U)     []    I

    subsetrefl(S,U) = subsetintro(S,U,U,[a,h]h)


subsettrans : (S:Set;
               U:(S)Set;
               V:(S)Set;
               W:(S)Set;
               subset(S,U,V);
               subset(S,V,W))
                 subset(S,U,W)     []    I

    subsettrans(S,U,V,W,subsetintro(_,_,_,h2),subsetintro(_,_,_,h)) =
        subsetintro(S,U,W,[a,h1]h(a,h2(a,h1)))
```

B Formal topology

```
COV : (S:Set;cov:(S;(S)Set)Set;U:(S)Set;V:(S)Set)Set     []    C

COVintro : (S:Set;
            cov:(S;(S)Set)Set;
            U:(S)Set;
            V:(S)Set;
            (a:S;U(a))cov(a,V))
              COV(S,cov,U,V)     []    C
```

```
MEET : (S:Set;meet:(S;S)S;U:(S)Set;V:(S)Set;S)Set     []    C

MEETintro : (S:Set;
             meet:(S;S)S;
             U:(S)Set;
             V:(S)Set;
             a:S;
             b:S;
             U(a);
             V(b))
               MEET(S,meet,U,V,meet(a,b))     []    C


POS : (S:Set;pos:(S)Set;U:(S)Set)Set     []    C

POSintro : (S:Set;
            pos:(S)Set;
            U:(S)Set;
            b:S;
            U(b);
            pos(b))
              POS(S,pos,U)     []    C


Sing = Id : (S:Set;S;S)Set     []


TOP is [S:Set; one:S; meet:(S;S)S; cov:(S;(S)Set)Set; pos:(S)Set;
        covmeet1:(a:S)cov(a,Sing(S,one));
        covrefl:(a:S;U:(S)Set;U(a))cov(a,U);
        covtrans:(a:S;U:(S)Set;V:(S)Set;cov(a,U);f:COV(S,cov,U,V))cov(a,V);
        covmeetl1:(a:S;b:S;U:(S)Set;cov(a,U))cov(meet(a,b),U);
        covmeetl2:(a:S;b:S;U:(S)Set;cov(b,U))cov(meet(a,b),U);
        covmeetr:(a:S;
                  U:(S)Set;
                  V:(S)Set;
                  cov(a,U);
                  cov(a,V))
                    cov(a,MEET(S,meet,U,V));
        mono:(a:S;U:(S)Set;pos(a);cov(a,U))POS(S,pos,U);
        posi:(a:S;U:(S)Set;(pos(a))cov(a,U))cov(a,U)]
```

C A concrete topology: Neighbourhoods of the natural numbers

```
SN : Set     []    C

onenat : SN     []    C
zero : SN     []    C
s : (SN)SN     []    C
ff : SN     []    C


meetnat : (a:SN;b:SN)SN     []    I
        meetnat(onenat,b) = b
        meetnat(zero,onenat) = zero
        meetnat(zero,zero) = zero
        meetnat(zero,s(h)) = ff
        meetnat(zero,ff) = ff
        meetnat(s(h),onenat) = s(h)
        meetnat(s(h),zero) = ff
        meetnat(s(h),s(h1)) = s(meetnat(h,h1))
        meetnat(s(h),ff) = ff
        meetnat(ff,b) = ff


leqnat = [a,b]Id(SN,meetnat(a,b),a) : (a:SN;b:SN)Set     []


posnat : (a:SN)Set     []    I
        posnat(onenat) = N1
        posnat(zero) = N1
        posnat(s(h)) = posnat(h)
        posnat(ff) = Empty


covnat : (a:SN;U:(SN)Set)Set     []    C

covnati1 : (a:SN;U:(SN)Set;(posnat(a))Empty)covnat(a,U)     []    C
covnati2 : (a:SN;U:(SN)Set;b:SN;U(b);leqnat(a,b))covnat(a,U)     []    C
```

We present only the types, because of the length of the proofs:

```
covmeetnat1 : (a:SN)covnat(a,Sing(SN,onenat))     []

covreflnat : (a:SN;U:(SN)Set;U(a))covnat(a,U)     []

covmeetnatl1 : (a:SN;b:SN;U:(SN)Set;covnat(a,U))covnat(meetnat(a,b),U)     []
```

```
covmeetnatl2 : (a:SN;b:SN;U:(SN)Set;covnat(b,U))covnat(meetnat(a,b),U)     []

covmeetnatr : (a:SN;
               U:(SN)Set;
               V:(SN)Set;
               covnat(a,U);
               covnat(a,V))
                 covnat(a,MEET(SN,meetnat,U,V))     []

covtransnat : (a:SN;
               U:(SN)Set;
               V:(SN)Set;
               covnat(a,U);
               f:COV(SN,covnat,U,V))
                 covnat(a,V)     []

mononat : (a:SN;U:(SN)Set;posnat(a);covnat(a,U))POS(SN,posnat,U)     []

posinat : (a:SN;U:(SN)Set;(posnat(a))covnat(a,U))covnat(a,U)     []


TOPNAT is {S:=SN; one:=onenat; meet:=meetnat; cov:=covnat; pos:=posnat;
           covmeet1:=covmeetnat1; covrefl:=covreflnat;
           covtrans:=covtransnat; covmeetl1:=covmeetnatl1;
           covmeetl2:=covmeetnatl2; covmeetr:=covmeetnatr;
           mono:=mononat;posi:=posinat} : TOP     []
```

D Semilattice

Order between the formal neighbourhoods:

```
leq = [a,b]cov(a,Sing(S,b)) : (a:S;b:S)Set     TOP
```

Equality between the formal neighbourhoods:

```
eqs = [a,b]Product(leq(a,b),leq(b,a)) : (S;S)Set     TOP
```

$\langle$`S,meet,one`$\rangle$ with order `leq` form a semilattice. We only present the types, since the proofs are too long:

```
meetcomm : (a:S;b:S)eqs(meet(a,b),meet(b,a))     TOP

meetassoc : (a:S;b:S;c:S)eqs(meet(a,meet(b,c)),meet(meet(a,b),c))     TOP

meetunit : (a:S)eqs(meet(one,a),a)     TOP

meetidem : (a:S)eqs(meet(a,a),a)     TOP
```

E A formal topology defines a frame/complete Heyting algebra

Equivalence between subsets:

```
EQS = [U,V]Product(COV(S,cov,U,V),COV(S,cov,V,U)) : (U:(S)Set;V:(S)Set)Set     TOP
```

Join in the formal topology:

```
JOIN = [T,I,U]union(S,T,I,U) : (T:Set;I:(T)Set;U:(T;S)Set;S)Set     TOP
```

(`union` is defined in appendix K).

Implication in complete Heyting algebra:

```
cHaimply = [U,V,a]COV(S,cov,MEET(S,meet,U,Sing(S,a)),V) :
           (U:(S)Set;V:(S)Set;a:S)Set     TOP
```

A formal topology defines a frame/complete Heyting algebra. We only present the types, because of the length of the proof terms.

`EQS` is an equivalence relation:

```
EQSrefl : (U:(S)Set)EQS(U,U)     TOP

EQSsymm : (U:(S)Set;V:(S)Set;EQS(U,V))EQS(V,U)     TOP

EQStrans : (U:(S)Set;V:(S)Set;W:(S)Set;EQS(U,V);EQS(V,W))EQS(U,W)     TOP
```

`COV` is a partial order on the subsets of `S` (antisymmetri is direct from the definition of `EQS`):

```
COVrefl : (U:(S)Set)COV(S,cov,U,U)     TOP

COVtrans : (U:(S)Set;
            V:(S)Set;
            W:(S)Set;
            COV(S,cov,U,V);
            COV(S,cov,V,W))
              COV(S,cov,U,W)     TOP
```

`COV` respects `EQS`:

```
COVrespEQS : (U:(S)Set;
              V:(S)Set;
              U':(S)Set;
              V':(S)Set;
              COV(S,cov,U,V);
              EQS(U,U');
              EQS(V,V'))
                COV(S,cov,U',V')     TOP
```

`EQS` respects `MEET`:

```
EQSrespMEET : (U:(S)Set;
               U':(S)Set;
               V:(S)Set;
               V':(S)Set;
               EQS(U,U');
               EQS(V,V'))
                 EQS(MEET(S,meet,U,V),MEET(S,meet,U',V'))     TOP
```

`EQS` respects `JOIN`:

```
EQSrespJOIN : (T:Set;
               I:(T)Set;
               U:(T;S)Set;
               U':(T;S)Set;
               (i:T;I(i))EQS(U(i),U'(i)))
                 EQS(JOIN(T,I,U),JOIN(T,I,U'))     TOP
```

The following proofs show that `MEET`, `JOIN` and `cHaimply` have the correct properties:

```
MEETisinfl1 : (U:(S)Set;V:(S)Set)COV(S,cov,MEET(S,meet,U,V),U)     TOP

MEETisinfl2 : (U:(S)Set;V:(S)Set)COV(S,cov,MEET(S,meet,U,V),V)     TOP

MEETisinfr : (W:(S)Set;
              U:(S)Set;
              V:(S)Set;
              COV(S,cov,W,U);
              COV(S,cov,W,V))
                COV(S,cov,W,MEET(S,meet,U,V))     TOP

MEETempty : (U:(S)Set)COV(S,cov,U,Sing(S,one))     TOP

JOINissup1 : (T:Set;I:(T)Set;U:(T;S)Set;i:T;I(i))
               COV(S,cov,U(i),JOIN(T,I,U))     TOP

JOINissup2 : (T:Set;
              I:(T)Set;
              U:(T;S)Set;
              V:(S)Set;
              (i:T;I(i))COV(S,cov,U(i),V))
                COV(S,cov,JOIN(T,I,U),V)     TOP

infdistr : (T:Set;
            I:(T)Set;
            V:(S)Set;
            U:(T;S)Set)EQS(MEET(S,meet,V,JOIN(T,I,U)),
                           JOIN(T,I,[i]MEET(S,meet,V,U(i))))     TOP
```

```
cHaimplyrespEQS : (U:(S)Set;
                   U':(S)Set;
                   V:(S)Set;
                   V':(S)Set;
                   EQS(U,U');
                   EQS(V,V'))
                     EQS(cHaimply(U,V),cHaimply(U',V'))     TOP

cHAimplyprop1 : (W:(S)Set;
                 U:(S)Set;
                 V:(S)Set;
                 COV(S,cov,W,cHaimply(U,V)))
                   COV(S,cov,MEET(S,meet,W,U),V)     TOP

cHaimplyprop2 : (W:(S)Set;
                 U:(S)Set;
                 V:(S)Set;
                 COV(S,cov,MEET(S,meet,W,U),V))
                   COV(S,cov,W,cHaimply(U,V))     TOP
```

F Closure operator

Closure operator:

```
Cl = [U,a]cov(a,U) : (U:(S)Set;a:S)Set     TOP
```

Meet for the closed sets:

```
MEETsat = [U,V,a]Product(U(a),V(a)) : (U:(S)Set;V:(S)Set;a:S)Set     TOP
```

Join for the closed sets:

```
JOINsat = [T,I,U]Cl(JOIN(T,I,U)) : (T:Set;I:(T)Set;U:(T;S)Set;S)Set     TOP
```

Predicate for closed sets:

```
sat = [U]eqsubset(S,U,Cl(U)) : (U:(S)Set)Set     TOP
```

We only present the types, because of the length of the proofs.

The covering order between closed subsets is the subset order:

```
Cllemma1a : (U:(S)Set;V:(S)Set;COV(S,cov,U,V))subset(S,U,Cl(V))     TOP

Cllemma1b : (U:(S)Set;V:(S)Set;subset(S,U,Cl(V)))COV(S,cov,U,V)     TOP
```

Each equivalence class contains exactly one closed subset:

```
Cllemma2 : (U:(S)Set)EQS(U,Cl(U))     TOP

Cllemma3 : (U:(S)Set;V:(S)Set;EQS(U,V))eqsubset(S,Cl(U),Cl(V))     TOP
```

`Cl` is a closure operator:

```
Clprop1 : (U:(S)Set)subset(S,U,Cl(U))     TOP

Clprop2 : (U:(S)Set;V:(S)Set;subset(S,U,V))subset(S,Cl(U),Cl(V))     TOP

Clprop3 : (U:(S)Set)eqsubset(S,Cl(Cl(U)),Cl(U))     TOP
```

The closed subsets form a frame which is isomorphic to the frame formed by the equivalence classes and `Cl` is a cHa isomorphism:

```
ClJOIN2JOINsat : (T:Set;
                  I:(T)Set;
                  U:(T;S)Set)
                    eqsubset(S,
                             Cl(JOIN(T,I,U)),
                             JOINsat(T,I,[i]Cl(U(i))))     TOP

ClMEET2MEETsatempty : (a:S)Cl(Sing(S,one),a)     TOP

ClMEET2MEETsatbin : (U:(S)Set;
                     V:(S)Set)
                       eqsubset(S,
                                Cl(MEET(S,meet,U,V)),
                                MEETsat(Cl(U),Cl(V)))     TOP

satcHaimply : (U:(S)Set;V:(S)Set)sat(cHaimply(U,V))     TOP

ClprescHaimply : (U:(S)Set;
                  V:(S)Set)
                    eqsubset(S,Cl(cHaimply(U,V)),cHaimply(Cl(U),Cl(V)))     TOP
```

G Points of a formal topology

`P` is a function that given a point returns a completely prime filter:

```
P : (p:(S)Set;U:(S)Set)Set     TOP     C

Pintro : (p:(S)Set;U:(S)Set;b:S;U(b);p(b))P(p,U)     TOP     C
```

We only present the types, because of the length of the proof terms.

Any point defines a completely prime filter:

```
point2filter1 : (p:(S)Set;
                 p(one);
                 (a:S;b:S;p(a);p(b))p(meet(a,b));
                 (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
                 (a:S;p(a))pos(a);
                 U:(S)Set;
                 V:(S)Set;
                 COV(S,cov,U,V);
                 P(p,U))
                   P(p,V)     TOP

point2filter2 : (p:(S)Set;
                 p(one);
                 (a:S;b:S;p(a);p(b))p(meet(a,b));
                 (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
                 (a:S;p(a))pos(a))
                   P(p,Sing(S,one))     TOP

point2filter3 : (p:(S)Set;
                 p(one);
                 (a:S;b:S;p(a);p(b))p(meet(a,b));
                 (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
                 (a:S;p(a))pos(a);
                 U:(S)Set;
                 V:(S)Set;
                 P(p,U);
                 P(p,V))
                   P(p,MEET(S,meet,U,V))     TOP

point2filter4 : (p:(S)Set;
                 p(one);
                 (a:S;b:S;p(a);p(b))p(meet(a,b));
                 (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
                 (a:S;p(a))pos(a);
                 T:Set;
                 I:(T)Set;
                 U:(T;S)Set;
                 P(p,JOIN(T,I,U)))
                   Exists(T,[i]Product(I(i),P(p,U(i))))     TOP
```

```
point2filter5 : (p:(S)Set;
                 p(one);
                 (a:S;b:S;p(a);p(b))p(meet(a,b));
                 (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
                 (a:S;p(a))pos(a);
                 U:(S)Set;
                 P(p,U))
                   POS(S,pos,U)     TOP
```

Any completely prime filter defines a point:

```
filter2point1 : (F:((S)Set)Set;
                 (U:(S)Set;V:(S)Set;COV(S,cov,U,V);F(U))F(V);
                 F(Sing(S,one));
                 (U:(S)Set;V:(S)Set;F(U);F(V))F(MEET(S,meet,U,V));
                 (T:Set;I:(T)Set;U:(T;S)Set;F(JOIN(T,I,U)))
                   Exists(T,[i]Product(I(i),F(U(i))));
                 (U:(S)Set;F(U))POS(S,pos,U))
                   F(Sing(S,one))     TOP

filter2point2 : (F:((S)Set)Set;
                 (U:(S)Set;V:(S)Set;COV(S,cov,U,V);F(U))F(V);
                 F(Sing(S,one));
                 (U:(S)Set;V:(S)Set;F(U);F(V))F(MEET(S,meet,U,V));
                 (T:Set;I:(T)Set;U:(T;S)Set;F(JOIN(T,I,U)))
                   Exists(T,[i]Product(I(i),F(U(i))));
                 (U:(S)Set;F(U))POS(S,pos,U);
                 a:S;
                 b:S;
                 F(Sing(S,a));
                 F(Sing(S,b)))
                   F(Sing(S,meet(a,b)))     TOP

filter2point3 : (F:((S)Set)Set;
                 (U:(S)Set;V:(S)Set;COV(S,cov,U,V);F(U))F(V);
                 F(Sing(S,one));
                 (U:(S)Set;V:(S)Set;F(U);F(V))F(MEET(S,meet,U,V));
                 (T:Set;I:(T)Set;U:(T;S)Set;F(JOIN(T,I,U)))
                   Exists(T,[i]Product(I(i),F(U(i))));
                 (U:(S)Set;F(U))POS(S,pos,U);
                 a:S;
                 U:(S)Set;
                 cov(a,U);
                 F(Sing(S,a)))
                   P([x]F(Sing(S,x)),U)     TOP
```

```
filter2point4 : (F:((S)Set)Set;
                 (U:(S)Set;V:(S)Set;COV(S,cov,U,V);F(U))F(V);
                 F(Sing(S,one));
                 (U:(S)Set;V:(S)Set;F(U);F(V))F(MEET(S,meet,U,V));
                 (T:Set;I:(T)Set;U:(T;S)Set;F(JOIN(T,I,U)))
                   Exists(T,[i]Product(I(i),F(U(i))));
                 (U:(S)Set;F(U))POS(S,pos,U);
                 a:S;
                 F(Sing(S,a)))
                   pos(a)     TOP
```

P is a bijection:

```
pfbij1a : (p:(S)Set;
           p(one);
           (a:S;b:S;p(a);p(b))p(meet(a,b));
           (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
           (a:S;p(a))pos(a);
           a:S;
           p(a))
             P(p,Sing(S,a))     TOP

pfbij1b : (p:(S)Set;
           p(one);
           (a:S;b:S;p(a);p(b))p(meet(a,b));
           (a:S;U:(S)Set;p(a);cov(a,U))P(p,U);
           (a:S;p(a))pos(a);
           a:S;
           P(p,Sing(S,a)))
             p(a)     TOP

pfbij2a : (F:((S)Set)Set;
           (U:(S)Set;V:(S)Set;COV(S,cov,U,V);F(U))F(V);
           F(Sing(S,one));
           (U:(S)Set;V:(S)Set;F(U);F(V))F(MEET(S,meet,U,V));
           (T:Set;I:(T)Set;U:(T;S)Set;F(JOIN(T,I,U)))
             Exists(T,[i]Product(I(i),F(U(i))));
           (U:(S)Set;F(U))POS(S,pos,U);
           U:(S)Set;
           F(U))
             P([a]F(Sing(S,a)),U)     TOP
```

```
pfbij2b : (F:((S)Set)Set;
           (U:(S)Set;V:(S)Set;COV(S,cov,U,V);F(U))F(V);
           F(Sing(S,one));
           (U:(S)Set;V:(S)Set;F(U);F(V))F(MEET(S,meet,U,V));
           (T:Set;I:(T)Set;U:(T;S)Set;F(JOIN(T,I,U)))
             Exists(T,[i]Product(I(i),F(U(i))));
           (U:(S)Set;F(U))POS(S,pos,U);
           U:(S)Set;
           P([a]F(Sing(S,a)),U))
             F(U)    TOP
```

H Scott topology

```
SCOTTOP is TOP + [scott:(a:S;
                         U:(S)Set;
                         cov(a,U);
                         pos(a))
                           Exists(S,[b]Product(U(b),cov(a,Sing(S,b))))]

SCOTTOP1 is SCOTTOP + [pos1:pos(one)]
```

I A concrete Scott topology: Neighbourhoods of the natural numbers

Here the example from appendix C continues.

```
scottnat : (a:SN;
            U:(SN)Set;
            covnat(a,U);
            posnat(a))
              Exists(SN,[b]Product(U(b),covnat(a,Sing(SN,b))))     []    I

    scottnat(a,U,covnati1(_,_,h2),h1) =
        case0([h]Exists(SN,[b]Product(U(b),covnat(a,Sing(SN,b)))),h2(h1))

    scottnat(a,U,covnati2(_,_,b,h2,h3),h1) =
        Exists_intro(SN,
                     [b']Product(U(b'),covnat(a,Sing(SN,b'))),
                     b,
                     pair(U(b),
                          covnat(a,Sing(SN,b)),
                          h2,
                          covnati2(a,Sing(SN,b),b,id(SN,b),h3)))
```

```
TOPNAT2 is {S:=SN; one:=onenat; meet:=meetnat; cov:=covnat; pos:=posnat;
            covmeet1:=covmeetnat1; covrefl:=covreflnat; covtrans:=covtransnat;
            covmeetl1:=covmeetnatl1; covmeetl2:=covmeetnatl2;
            covmeetr:=covmeetnatr; mono:=mononat; posi:=posinat;
            scott:=scottnat} : SCOTTOP     []
```

J Points of a Scott topology

Points of a Scott topology:

```
scpoint : (p:(S)Set)Set     SCOTTOP    C

scpintro : (p:(S)Set;
           p(one);
           (a:S;b:S;p(a);p(b))p(meet(a,b));
           (a:S;b:S;p(a);leq(a,b))p(b);
           (a:S;p(a))pos(a))
             scpoint(p)     SCOTTOP    C
```

In a Scott topology, scpoint is the same as point:

```
point2scpoint = [p,h,h1,h2,h3]
    scpintro(p,
             h,
             h1,
             [a,b,h4,h5]Exists_elim(S,
                                    [x]Product(Sing(S,b,x),
                                               p(x)),
                                    [h6]p(b),
                                    [a',b']idsubst'(S,
                                                    [b1]p(b1),
                                                    b,
                                                    a',
                                                    proj1(Id(S,b,a'),p(a'),b'),
                                                    proj2(Sing(S,b,a'),p(a'),b')),
                                    h2(a,Sing(S,b),h4,h5)),
             h3) :
    (p:(S)Set;
     p(one);
     (a:S;b:S;p(a);p(b))p(meet(a,b));
     (a:S;U:(S)Set;p(a);cov(a,U))Exists(S,[x]Product(U(x),p(x)));
     (a:S;p(a))pos(a))
       scpoint(p)     SCOTTOP    I
```

```
scpoint2point : (p:(S)Set;
                 scpoint(p);
                 a:S;
                 U:(S)Set;
                 p(a);
                 cov(a,U))
                   Exists(S,[h]Product(U(h),p(h)))     SCOTTOP    I
    scpoint2point(p,scpintro(_,h3,h4,h5,h6),a,U,h1,h2) =
      Exists_elim(S,
                  [b]Product(U(b),cov(a,Sing(S,b))),
                  [z]Exists(S,[h]Product(U(h),p(h))),
                  [a',b]Exists_intro(S,
                                     [h]Product(U(h),p(h)),
                                     a',
                                     pair(U(a'),
                                          p(a'),
                                          proj1(U(a'),cov(a,Sing(S,a')),b),
                                          h5(a,a',h1,proj2(U(a'),leq(a,a'),b)))),
                  scott(a,U,h2,h6(a,h1)))
```

K Union of subsets and directed families

Union of subsets, where the index set itself is a subset:

```
union : (S:Set;T:Set;I:(T)Set;U:(T;S)Set;S)Set     []    C

unionintro : (S:Set;
              T:Set;
              I:(T)Set;
              U:(T;S)Set;
              i:T;
              I(i);
              a:S;
              U(i,a))
                union(S,T,I,U,a)     []    C
```

Directed families, where the index set is a subset:

```
directed : (S:Set;T:Set;I:(T)Set;p:(T;S)Set)Set     []    C

directedintro : (S:Set;
                 T:Set;
                 I:(T)Set;
                 p:(T;S)Set;
                 i0:T;
                 I(i0);
                 (i:T;j:T;I(i);I(j))
                   Exists(T,[k]Product(I(k),
                                       Product(subset(S,p(i),p(k)),
                                               subset(S,p(j),p(k))))))
                     directed(S,T,I,p)     []    C
```

L The points in a Scott topology form a Scott domain

We only present the types, since the proofs are too long.

The points in a Scott topology form a cpo:

```
leqonemin : (p:(S)Set;scpoint(p))subset(S,leq(one),p)     SCOTTOP

unionpoint : (T:Set;
              I:(T)Set;
              p:(T;S)Set;
              (i:T;I(i))scpoint(p(i));
              directed(S,T,I,p))
                scpoint(union(S,T,I,p))     SCOTTOP

unionsup1 : (T:Set;
             I:(T)Set;
             p:(T;S)Set;
             i:T;
             I(i))
               subset(S,p(i),union(S,T,I,p))     TOP

unionsup2 : (T:Set;
             I:(T)Set;
             p:(T;S)Set;
             q:(S)Set;
             (i:T;I(i))subset(S,p(i),q))
               subset(S,union(S,T,I,p),q)     TOP
```

which is algebraic and every two points has a least upper bound:

```
genscp : (a:S;pos(a))scpoint(leq(a))     SCOTTOP
```

```
genscpdir : (p:(S)Set;scpoint(p))directed(S,S,p,leq)     SCOTTOP

gencompscp : (p:(S)Set;
              scpoint(p);
              (T:Set;
               I:(T)Set;
               p2:(T;S)Set;
               directed(S,T,I,p2);
               subset(S,p,union(S,T,I,p2)))
                 Exists(T,[h]Product(I(h),subset(S,p,p2(h)))))
                   Exists(S,[h]Product(p(h),eqsubset(S,p,leq(h))))     SCOTTOP

scpcomplb1 : (q:(S)Set;
              scpoint(q);
              (T:Set;
               I:(T)Set;
               r:(T;S)Set;
               directed(S,T,I,r);
               subset(S,q,union(S,T,I,r)))
                 Exists(T,[x]Product(I(x),subset(S,q,r(x))));
              p:(S)Set;
              subset(S,q,p))
                Exists(S,[a]Product(p(a),eqsubset(S,q,leq(a))))     SCOTTOP

scpcomplb2a : (p:(S)Set;
               q:(S)Set;
               scpoint(p);
               Exists(S,[a]Product(p(a),eqsubset(S,q,leq(a)))))
                 subset(S,q,p)     SCOTTOP

scpcomplb2b : (p:(S)Set;
               q:(S)Set;
               Exists(S,[a]Product(p(a),eqsubset(S,q,leq(a))));
               T:Set;
               I:(T)Set;
               r:(T;S)Set;
               (i:T;I(i))scpoint(r(i));
               subset(S,q,union(S,T,I,r)))
                 Exists(T,[x]Product(I(x),subset(S,q,r(x))))     SCOTTOP

supcompscp : (p:(S)Set;scpoint(p))eqsubset(S,p,union(S,S,p,leq))     SCOTTOP
```

```
psuptoleqp : (p1:(S)Set;
              p2:(S)Set;
              scpoint(p1);
              scpoint(p2);
              (T:Set;
               I:(T)Set;
               q:(T;S)Set;
               directed(S,T,I,q);
               subset(S,p1,union(S,T,I,q)))
                 Exists(T,[h]Product(I(h),subset(S,p1,q(h))));
              (T:Set;
               I:(T)Set;
               q:(T;S)Set;
               directed(S,T,I,q);
               subset(S,p2,union(S,T,I,q)))
                 Exists(T,[h]Product(I(h),subset(S,p2,q(h))));
              r:(S)Set;
              scpoint(r);
              subset(S,p1,r);
              subset(S,p2,r);
              q:(S)Set;
              scpoint(q))
                Exists(S,[x]Product(Product(scpoint(leq(x)),
                                            Product(subset(S,p1,leq(x)),
                                                    subset(S,p2,leq(x)))),
                                    Imply(Product(subset(S,p1,q),
                                                  subset(S,p2,q)),
                                          subset(S,leq(x),q))))     SCOTTOP
```