Introduction to Formal Notations

Asymetrix Report 94-0

H.A. Proper
Asymetrix Research Laboratory
Department of Computer Science
University of Queensland
Australia 4072
E.Proper@acm.org

Version of June 23, 2004 at 10:29

PUBLISHED AS:

H.A. Proper. Introduction to formal notations. Asymetrix Research Report 94-0, Asymetrix Research Laboratory, University of Queensland, Brisbane, Australia, 1994.

Abstract

In this report we provide a short discussion on the formal notations used in the reports from the research lab. The intention is that this documents evolves in the course of time. This evolution is driven by two sources. Firstly, any unclarities in the used notations, either due to terse explanation or limitations in the existing knowledge in the fields of formalisation. Secondly, due to the possible introduction of new notations and formalisms in new reports.

Although this first report is entirely written by me, it is the intention that all researches from the research lab, so: T.A. Halpin and L. Campbell, will contribute to this report when needed.

1 Sets

A set itself is either denoted by enumeration, such as: $\{1, 2, 3\}$, $\{10, \ldots, 100\}$, or by a using a so called set comprehension:

$$\{x \mid P(x)\}$$

The occurrence of an element x in a set X is denoted as $x \in X$. By $x \notin X$ we denote the fact that x does *not* occur in X. As an example set comprehension, the following set contains all positive even numbers:

$$\{n \mid n \in \mathbb{N} \wedge (n \,\mathsf{DIV}\, 2) \times 2 = n\}$$

Alternatively, this can be rewritten as:

$$\{n \in \mathbb{N} \mid (n \,\mathsf{DIV}\, 2) \times 2 = n\}$$

The empty set, i.e. the set that does not contain any value, is denoted as: $\varnothing$.

In definitions of sets or values in general, we will usually use the format $T \triangleq E$, where T is the term to be defined and E is the defining expressions. On sets we can now define the following additional operations:

$$\begin{array}{rcll}
A \subseteq B & \iff & \forall_x\, [x \in A \Rightarrow x \in B] & \text{subset} \\
A \subset B & \iff & A \subseteq B \wedge A \neq B & \text{proper subset} \\
A \not\subseteq B & \iff & \neg A \subseteq B & \text{no subset} \\
A \not\subset B & \iff & \neg A \subset B & \text{no proper subset} \\
A \cup B & \triangleq & \{x \mid x \in A \vee x \in B\} & \text{union} \\
A \cap B & \triangleq & \{x \mid x \in A \wedge x \in B\} & \text{intersection} \\
A - B & \triangleq & \{x \mid x \in A \wedge x \notin B\} & \text{difference/filtering} \\
\wp(A) & \triangleq & \{Y \mid Y \subseteq X\} & \text{powerset}
\end{array}$$

For the $\wp$ operator it is important to realise that the emptyset is always included, and that the result is a set of sets. For instance, $\wp(\{1,2\}) = \{\{1,2\},\{1\},\{2\},\varnothing\}$. Related to this operation is the following variation of the union operation. If X is a set of sets, then we can define: $\cup X \triangleq \{x \mid \exists_{A \in X}\, [x \in A]\}$, which unites all sets in X into one large set. As an example: $\cup \{\{1,2\},\{a,b,c\}\} = \{1,2,a,b,c\}$.

The number of elements in a set X is counted by $|X|$. So we have: $|\{1,2,3,a,b\}| = 5$.

Some sets will have a total ordering or arithmetic operations defined over them, for instance $\mathbb{N}$. For such sets, the following operations are usefull:

$$\begin{array}{rcl}
\mathsf{max}(X) & \triangleq & x \in X \textbf{ such that } \forall_{a \in X}\, [a \leq x] \\
\mathsf{min}(X) & \triangleq & x \in X \textbf{ such that } \forall_{a \in X}\, [x \geq a] \\
\Sigma(\{x_1, \ldots, x_n\}) & \triangleq & x_1 + \ldots + x_n
\end{array}$$

Some further notational convention concerning these operations is:

$$\begin{array}{rcl}
\mathsf{max}_{x \in X}\, E(x) & \triangleq & \mathsf{max}(\{E(x) \mid x \in X\}) \\
\mathsf{min}_{x \in X}\, E(x) & \triangleq & \mathsf{min}(\{E(x) \mid x \in X\}) \\
\Sigma_{x \in X} E(x) & \triangleq & E(x_1) + \ldots + E(x_n) \text{ if } X = \{x_1, \ldots, x_n\}
\end{array}$$

where $E(x)$ is an expression in terms x.

The operations on sets as defined above are quite uniform, and can be found in most literature on set theory. However, we introduce two special operations that are not common to the literature, but that allow us to delete or add one element from a set:

$$\begin{aligned} A \ominus x &\triangleq A - \{x\} \\ X \oplus x &\triangleq A \cup \{x\} \end{aligned}$$

2 Tuples

Using the so-called cartesian product, we can introduce tuples. A tuple over domains $A_1, \ldots A_n$ is introduced as: $\langle a_1, \ldots, a_n \rangle \in A_1 \times \ldots \times A_n$. As an example, for $\{1,2\} \times \{a,b\}$ we have the following possible tuples: $\{\langle 1,a \rangle, \langle 1,b \rangle, \langle 2,a \rangle, \langle 2,b \rangle\}$.

In the literature no real standard operations on tuples exist. However, we introduce two operations on tuples. Tuples can be joined into larger tuples by:

$$\langle x_1, \ldots, x_n \rangle * \langle y_1, \ldots, y_n \rangle \triangleq \langle x_1, \ldots, x_n, y_1, \ldots, y_n \rangle$$

One element of a tuple can be selected by the *projection* operation: π_i. Formally:

$$\pi_i \ \langle a_1, \ldots, a_n \rangle \triangleq a_i$$

An example is: $\pi_2 \ \langle 1, a \rangle = a$. The projection operation is usually used in combination with sets of tuples rather than single tuples. For this purpose we generalise the projection operation as follows: $\pi_i \ X \triangleq \{\pi_i x \mid x \in X\}$, where X must be a set of tuples.

3 Relations

Now that we know what tuples are we can introduce relationships. An n-ary relationship over domains $A_1, \ldots, A_n$ is a set of tuples over $A_1 \times \ldots \times A_n$. An n-ary relationship R is therefore usually introduced as: $R \subseteq A_1 \times \ldots \times A_n$.

For a relationship R we have the following abbreviations:

$$\begin{aligned} R(x_1, \ldots, x_n) &\triangleq \langle x_1, \ldots, x_n \rangle \in R \\ x \, R \, y &\triangleq \langle x, y \rangle \in R \ \text{ if } R \text{ is a binary relationship} \end{aligned}$$

These abbreviations are both borrowed from predicate logic, and allow for elegant notations. The $x \ R \ y$ notation is a so-called *infix* notation. They are quite common. For instance: $\leq, <, =$ are all infix notations for binary relationships.

Binary relations can be combined into new binary relationships using the $\circ$ operations:

$$R \circ S \triangleq \{\langle x, z\rangle \mid \langle x, y\rangle \in R \wedge \langle y, z\rangle \in S\}$$

4 Functions

A partial function f from A to B is defined by $f : A \rightarrowtail B$. Formally, it is a relation $f \subseteq A \times B$ such that $\langle a, b\rangle \in f \wedge \langle a, c\rangle \in f \Rightarrow b = c$. A possibly more familiar verbalisation of this rule is: each A has at most one B. This property makes it possible to write $f(a) = b$ instead of $\langle a, b\rangle \in f$. Note that A and B could be complex objects. For instance: $f : (A \rightarrowtail B) \rightarrowtail C$ is a function that takes a function as its argument, and $f : A \times B \rightarrow C \times D \times E$ is a function that converts binary tuples to ternary tuples.

A total function is introduced by $f : A \rightarrow B$. From a formal point of view, it is a function $f : A \rightarrowtail B$ such that $a \in A \Rightarrow \exists_b [\langle a, b\rangle \in f]$, i.e. every A has a B.

A function f is a set of binary tuples. This allows for the following functions on functions:

$$\begin{array}{rcll} \mathsf{dom}(f) & \triangleq & \pi_1(f) & \text{the domain of a function} \\ \mathsf{ran}(f) & \triangleq & \pi_2(f) & \text{the range of a function} \end{array}$$

We obviously have for a function $f : A \rightarrowtail B$:

$$\mathsf{dom}(f) \subseteq A \text{ and } \mathsf{ran}(f) \subseteq B$$

For a total function $f : A \rightarrow B$ we have: $\mathsf{dom}(f) = A$.

The above definitions for functions are rather standard, and can be found in the literature. We introduce some more abbreviations for reasons of convenience:

$$\begin{array}{rcll} f(a)\downarrow & \triangleq & a \in \mathsf{dom}(f) & \text{function } f \text{ is defined for } a \\ f(a)\uparrow & \triangleq & a \notin \mathsf{dom}(f) & \text{function } f \text{ is \textit{not} defined for } a \end{array}$$

For unary functions, we will write $f\downarrow a$, and $f\uparrow a$, instead of $f(a)\downarrow$, and $f(a)\uparrow$ respectively. Evenmore, $f_1, \dots, f_n\downarrow a_1, \dots, a_m$ is employed as an abbreviation for: $\forall_{1\le i\le n}\forall_{1\le j\le m} [f_i\downarrow a_j]$. For a function f we also define the following operation which allows us to override existing values:

$$f \oslash \langle x, y\rangle \quad \triangleq \quad \left\{\langle v, w\rangle \in f \;\middle|\; v \neq x\right\} \oplus \langle x, y\rangle$$

This operation replaces (if existing) the value associated to x by y.

In some cases, functions have to be denoted as an expression, for this purpose we apply lambda calculus ([Bar84]). An example is: $f = \lambda x.$**if** $x = 0$ **then** 0 **else** $1/x$ **fi**. In the definition of functions, we will use $\perp$ to indicate that the function is not defined for a particular value. Sometimes we will also use **error** instead of $\bot$, to indicate that the function/operation leads to an error situation.

The inverse of a function, or relation, R is defined as: $f^{\leftarrow} \triangleq \{\langle y, x\rangle \mid \langle x, y\rangle \in f\}$. The inverse $f^{\leftarrow}$ of a function, is a function if f is an injection (and thus a bijection on $\mathsf{dom}(f)$). Functions f and g can be composed, if $\mathsf{ran}(g) \subseteq \mathsf{dom}(f)$ by $\cdot$ as follows: $f \cdot g \triangleq \lambda x. f(g(x))$.

5 Sequences

Sequences, or lists, are denoted as: $[x_1, \ldots, x_n]$. A sequence is basically a set with an order defined over it, where elements can occur more than once. The empty sequence is denoted as: $[\,]$. If X is a set of elements, than X^* is the set of all sequences of elements from X, and X^+ is the set of all non-empty sequences of elements from X. They are defined formally as:

$$\begin{aligned} X^+ &\triangleq \left\{ [x_1, \ldots, x_n] \;\middle|\; n < 0 \land \{x_1, \ldots, x_n\} \subseteq X \right\} \\ X^* &\triangleq X^+ \cup \{ [\,] \} \end{aligned}$$

Sequences are concatenated by the following operation:

$$[x_1, \ldots, x_n] \mathbin{+\!\!+} [y_1, \ldots, y_n] \triangleq [x_1, \ldots, x_n, y_1, \ldots, y_n]$$

We can select elements from a sequence $[x_1, \ldots, x_n]$ by the following operation:

$$[x_1, \ldots, x_n][i] \triangleq x_i$$

which results in the i-th element of the sequence. Similar to sets, we use the $\in$ operation to denote the occurrence of elements in a sequence S. It is defined as:

$$x \in S \iff \exists_i \, [S[i] = x]$$

Using the $\in$ operation for sequences, we can finally introduce a usefull operation coercing between a sequence S and a set:

$$\mathsf{SET}(S) \triangleq \left\{ s \;\middle|\; s \in S \right\}$$

6 Algorithms

As an example of the notations used for algorithms, consider the following algorithm:

```
GenPattern : $\mathcal{RL} \times (\mathcal{TP} \rightarrow \mathbb{N}) \rightarrow (\mathcal{TP} \rightarrowtail \mathbb{N}) \times \wp(\mathcal{RO} \rightarrow \mathbb{N})$
GenPattern$(Rel, Size) \triangleq$
 VAR
    Pattern:      $\wp(\mathcal{RO} \rightarrow \mathbb{N})$;
    FreshTuple:   $\mathcal{RO} \rightarrow \mathbb{N}$;
    WorkTuple:    $\mathcal{RO} \rightarrow \mathbb{N}$;
    Used:         $\mathcal{TP} \rightarrowtail \mathbb{N}$;
    $p$:            $\mathcal{RO}$;

 MACROS
   $Extendable(P : \wp(\mathcal{RO})) \equiv$
     $\forall_{p \in P}\, [Used(\mathsf{Player}(p)) < Size(\mathsf{Player}(p))] \wedge$
     $Used(\mathsf{Rel}(p)) + |P| \leq Size(\mathsf{Rel}(p))$;

   $IncrUsed(p : \mathcal{RO}) \equiv$
     BEGIN
       $Used(\mathsf{Player}(p)) \mathrel{+}:= 1$;
       $Used(\mathsf{Rel}(p)) \mathrel{+}:= 1$;
     END;

 BEGIN
   # Initialise variables
   $Pattern := \varnothing$;
   $Used(Rel) := 0$;
   FOR EACH $p \in \mathsf{Roles}(Rel)$ DO
     $Used(\mathsf{Player}(p)) := 0$;
   END FOR;

   WHILE $Extendable(\mathsf{Roles}(p))$ DO
     # Generate fresh tuple
     FOR EACH $p \in \mathsf{Roles}(Rel)$ DO
       $IncrUsed(p)$;
       $FreshTuple(p) := Used(\mathsf{Player}(p))$;
     END FOR;

     # Probe uniqueness
```

```
      $Pattern \mathrel{+}:= \{FreshTuple\}$;
      FOR EACH $p \in \mathsf{Roles}(Rel)$ DO
        $WorkTuple := FreshTuple$;

        # Try to mutate tuple
        IF $\neg\exists_{\tau \subseteq \mathsf{Roles}(Rel)} [\mathsf{Unique}(\tau) \land p \notin \tau] \land Extendable(\{p\})$ THEN
          $IncrUsed(p)$;
          $WorkTuple(p) := Used(\mathsf{Player}(p))$;
          $Pattern \mathrel{+}:= \{WorkTuple\}$;
        END IF;
      END FOR;

      # Generate nil tuple
      FOR EACH $p \in \mathsf{Roles}(Rel)$ DO
        $IncrUsed(p)$;
        $FreshTuple(p) := 0$;
      END FOR;

      # Probe totality
      FOR EACH $p \in \mathsf{Roles}(Rel)$ DO
        $WorkTuple := FreshTuple$;

        # Try to mutate tuple
        IF $\neg\,\mathsf{Total}(p) \land Extendable(\{p\})$ THEN
          $IncrUsed(p)$;
          $WorkTuple(p) := Used(\mathsf{Player}(p))$;
          $Pattern \mathrel{+}:= \{WorkTuple\}$;
        END IF;
      END FOR;
    END WHILE;

    RETURN $\langle Used, Pattern \rangle$;
  END.
```

Most aspects of this algorithm will be clear to people who are proficient with programming languages such as C or PASCAL. One important class of operations are the operations based on the $:=$ operation. The $:=$ operation is the traditional *becomes* statement from programming languages. For variables of simple types we now have the following abbreviations:

$$\begin{aligned} A \mathrel{+}:= B &\triangleq A := A + B \\ A \mathrel{-}:= B &\triangleq A := A - B \end{aligned}$$

For sets this becomes:

$$\begin{aligned} A \mathrel{+}:= B &\triangleq A := A \cup B \\ A \mathrel{-}:= B &\triangleq A := A - B \end{aligned}$$

In the case of a sequence we can write:

$$S \mathrel{+\!\!+}:= B \triangleq S := S \mathbin{+\!\!+} B$$

Finally, an important class of variables are functions. In an algorithm, they can be treated similarly to arrays. So if f is defined as a variable of type $A \rightarrow B$, then we would have:

$$f(A) := B \triangleq f := f \oslash \langle A, B \rangle$$

References

[Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, The Netherlands, Revised Edition, 1984.