

# Handel 0 report (draft)

## Introduction

Handel 0 is a simple programming language based upon Edsger Dijkstra's program notation.

This report serves as a reference for programmers, implementors and manual writers.

\*            \*  
              \*

## Syntax

In Handel we have *literals* and *identifiers*, which we combine to form *expressions*. Literals are simplest, so we will discuss them first.

A literal is either *boolean* or *natural*.

*literal* = *boolean* | *natural*

*boolean* = `true` | `false`

A natural literal is a sequence of digits, representing a number in base 10.

*digit* = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

*natural* = *digit*<sup>+</sup>

\*            \*  
              \*

An *identifier* in Handel is any sequence of letters and digits, beginning with a letter.

*ident* = *letter* (*letter* | *digit*)<sup>\*</sup>

Literals and identifiers are combined into *expressions* in the following way:

```

basicexpr = literal | ident | command | ( expr )
dotexpr = basicexpr [. basicexpr]
negation = (non |  $\sim$ )* dotexpr
factor = negation (* negation)*
term = factor ((+ | -) factor)*
relation = term [( = | <= | < | >= | > | =|) term]
disjunct = relation ((or | and) relation)*
assign = disjunct := disjunct
seq = assign(; assign)*
expr = seq

```

```

*           *
*

```

Commands are expressions which, when evaluated, change the state of the system. Such a state change is called a *side effect*. Handel 0 has three elementary commands and three composite ones. The elementary commands are **skip**, **abort** and the assignment. The composite commands are the block, alternative and repetitive constructs.

```

command = skip | abort | block | IF | DO
block = let decl(; decl)* in expr ni
decl = var ident:type
IF = if [guardedcmd(| guardedcmd)*] fi
DO = do [guardedcmd(| guardedcmd)*] od
guardedcmd = expr -> expr

```

```

*           *
*

```

Handel 0 supports three basic types: boolean, integer and array.

```

type = bool | int | arraytype
arraytype = array term of type

```

\*            \*  
              \*

## Semantics

In the sequel  $R$  is the predicate describing the state of the computation.

### **skip**

By definition,

$$[wp.\text{skip}.R \equiv R] \quad \text{for all } R .$$

### **abort**

By definition,

$$[wp.\text{abort}.R \equiv \perp] \quad \text{for all } R .$$

### **The assignment**

By definition,

$$[wp.(x := E).R \equiv (x := E).R] \quad \text{for all } R .$$

### **The sequential composition**

By definition,

$$[wp.(S0; S1).R \equiv wp.S0.(wp.S1.R)] \quad \text{for all } R .$$

In the sequel  $B.i \rightarrow S.i$  is the  $i$ th guarded statement.

### **The alternative construct**

By definition,

$$[wp.IF.R \equiv \langle \exists i :: B.i \rangle \wedge \langle \forall i : B.i : wp.(S.i).R \rangle] \text{ for all } R .$$

### The repetitive construct

By the *Invariance theorem for Repetitive Constructs*,

$$[P \Rightarrow wp.DO.(P \wedge \neg B)] \Leftarrow [\langle \forall i : P \wedge B.i : wp.(S.i).P \rangle]$$

\*            \*  
              \*  
              \*

### An example

```
let
  var N:int;
  var b: array N of bool;
  var x:int;
  var y:int
in
  x:=0; y:=N;
  do x =| y
    -> if non b.x -> x := x+1
       |      b.x -> y := x
    fi
  od
ni
```

2008.01.27

E. Emmanuel Macaulay  
eric@mathmeth.com