



50003

Models of Computation
Imperial College London

Contents

1	Introduction	2
1.1	Course Structure	2
1.2	Algorithms	2
1.3	Decision Problems	4
1.3.1	Hilbert's Entscheidungsproblem	4
1.4	Algorithms	4
1.4.1	Algorithms Informally	4
1.4.2	The Halting Problem	5
1.4.3	Algorithms as Functions	5
1.4.4	Haskell Programs	5
1.5	Program Semantics	6
2	While Language	7
2.1	SimpleExp	7
2.1.1	Big-Step Semantics	7
2.1.2	Small Step Semantics	8
2.2	While	9
2.2.1	Syntax	9
2.2.2	States	9
2.2.3	Rules	9
2.2.4	Properties	10
2.2.5	Configurations	10
2.2.6	Normalising	11
2.2.7	Side Effecting Expressions	11
2.2.8	Short Circuit Semantics	11
2.2.9	Strictness	11
2.2.10	Complex Programs	11
3	Credit	15

Chapter 1

Introduction

1.1 Course Structure



Dr Azelea Raad



Dr Herbert Wiklicky

First Half

- The while language
- Big & small step semantics
- Structural induction

Second Half

- Register Machines & gadgets
- Turing Machines
- Lambda Calculus

1.2 Algorithms

Euclid's Algorithm

Extra Fun! 1.2.1

Algorithm to find the greatest common divisor published by greek mathematician Euclid in ≈ 300 B.C.

```
-- continually take the modulus and compare until the modulus is zero
euclidGCD :: Int -> Int -> Int
euclidGCD a b
  | b == 0 = a
  | otherwise = euclidGCD b (a `mod` b)
```

Sieve of Eratosthenes

Extra Fun! 1.2.2

Used to find the prime numbers within a limit. Done by starting from the 2, adding the number to the primes, marking all multiples as non-prime, then repeating progressing to the next non-marked number (a prime) and repeating.

The sieve is attributed to Eratosthenes of Cyrene and was first published ≈ 200 B.C.

```
-- Filtering rather than marking elements
eraSieve :: Int -> [Int]
eraSieve lim = eraSieveHelper [2..lim]
  where
    eraSieveHelper :: [Int] -> [Int]
```

```
eraSieveHelper (x:xs) = x:eraSieveHelper (filter (\n -> n `mod` x /= 0) xs)
eraSieveHelper [] = []
```

Al-Khwarizmi

Extra Fun! 1.2.3

A persian polymath who first presented systematic solutions to linear and quadratic equations (by completing the square). He pioneered the treatment of algebra as an independent discipline within mathematics and introduced foundational methods such as the notion of balancing & reducing equal equations (e.g subtract/-cancel the same algebraic term from both sides of an equation)

His book title الجبر "*al-jabr*" resulted in the word *algebra* and subsequently algorithm.

Algorithms predate the computer, and have been studied in a mathematical/logical context for centuries.

- Very early attempts such as the Antikythera Mechanism (an analogue calculator for determining the positions of)
- Simple configurable machines (e.g automatic looms, pianola, census tabulating machines) invented in the 1800s.
- Basic calculation devices such as Charles Babbage's *Difference Engine* further generalised the idea of a calculating machine with a sequence of operations, and rudimentary memory store.
- Babbage's Analytical Engine is generally considered the world's first digital computer design, but was not fully implemented due to the limits of precision engineering at the time.
- English mathematician Ada Lovelace writes the first ever computer program (to calculate bernoulli numbers) on Babbage's analytical engine.

Note G

Extra Fun! 1.2.4

While translating a french transcript of a lecture given by Charles Babbage at the University of Turin on his analytical engine, Ada Lovelace added several notes (A-G), with the last including a description of an algorithm to compute the Bernoulli numbers.

Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.												Working Variables.												Result Variables.			
						v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}	v_{21}	v_{22}	v_{23}	v_{24}	v_{25}	v_{26}	v_{27}	
						1	2	n																									
1	$\times v_2 \times v_2$	$v_4 = v_2$			$-2n$		2	n	2n	2n	2n																						
2	$-v_4 - v_1$	$v_5 = v_4$			$-2n-1$	1																											
3	$+v_4 + v_1$	$v_6 = v_4$			$-2n+1$	1																											
4	$+v_4 - v_2$	$v_7 = v_4$			$\frac{2n-1}{2}$				0	0																							
5	$+v_4 + v_2$	$v_8 = v_4$			$\frac{1}{2} \cdot \frac{2n-1}{2}$		2																										
6	$-v_{10} - v_{11}$	$v_9 = v_{10}$			$\frac{1}{2} \cdot \frac{2n-1}{2} = A_0$																												
7	$-v_4 - v_1$	$v_{10} = v_4$			$n-1 (=3)$	1		n																									
8	$+v_2 + v_2$	$v_{11} = v_2$			$-2+0=2$		2																										
9	$+v_2 + v_2$	$v_{12} = v_2$			$\frac{2n}{2} = A_1$									2n	2																		
10	$\times v_{10} \times v_{11}$	$v_{13} = v_{10}$			$\frac{2n}{2} = A_1$																												
11	$+v_{10} + v_{11}$	$v_{14} = v_{10}$			$\frac{1}{2} \cdot \frac{2n-1}{2} + B_1 \cdot \frac{2n}{2}$																												
12	$-v_{10} - v_1$	$v_{15} = v_{10}$			$n-2 (=2)$	1																											
13	$-v_4 - v_1$	$v_{16} = v_4$			$-2n-1$	1																											
14	$+v_1 + v_2$	$v_{17} = v_1$			$-2+1=3$	1																											
15	$-v_4 + v_2$	$v_{18} = v_4$			$\frac{2n-1}{2}$																												
16	$\times v_2 \times v_{11}$	$v_{19} = v_2$			$\frac{2n}{2} = A_1$																												
17	$-v_4 - v_1$	$v_{20} = v_4$			$2n-2$	1																											
18	$+v_1 + v_2$	$v_{21} = v_1$			$-3+1=4$																												
19	$+v_2 + v_2$	$v_{22} = v_2$			$\frac{2n-2}{2}$																												
20	$\times v_2 \times v_{11}$	$v_{23} = v_2$			$\frac{2n}{2} = A_1$																												
21	$\times v_{10} \times v_{11}$	$v_{24} = v_{10}$			$\frac{2n}{2} = A_1$																												
22	$+v_{10} + v_{11}$	$v_{25} = v_{10}$			$A_0 + B_1 \cdot A_1 + B_2 \cdot A_2$																												
23	$-v_{10} - v_1$	$v_{26} = v_{10}$			$n-3 (=1)$	1																											

Here follows a repetition of Operations thirteen to twenty-three.

24	$+v_{10} + v_{11}$	$v_{27} = v_{10}$			$n+1 = 4+1=5$																											
25	$+v_1 + v_2$	$v_{28} = v_1$			by a Variable-card.	1		n+1																								
		$v_{29} = v_2$			by a Variable-card.																											

Babbage's Machines

Extra Fun! 1.2.5

The *Difference Engine* was used as the basis for designing the fully programmable *Analytical Engine*.

- Held back by lack of funds, limitations of precision machining at the time.
- Contains an ALU for arithmetic operations, supports conditional branches and has a memory

- Part of the machine (including a printing mechanism) are on display at the science museum.

1.3 Decision Problems

Formulas

Definition 1.3.1

Well formed logical statements that are a sequence of symbols form a given formal language. e.g $(p \vee q) \wedge i$ is a formula, but $) \vee \wedge ji$ is not.

Given:

- A set S of finite data structures of some kind (e.g formulae in first order logic).
- A property P of elements of S (e.g the property of a formula that it has a proof).

The associated decision procedure is:

Find an algorithm such that for any $s \in S$, if s has property P the algorithm terminates with 1, otherwise with 0.

1.3.1 Hilbert's Entscheidungsproblem

Is there an algorithm which can take any statement in first-order logic, and determine in a finite number of steps if the statement is provable?

First Order Logic/Predicate Logic

Definition 1.3.2

An extension of propositional logic that includes quantifiers (\forall, \exists), equality, function symbols (e.g $\times, \div, +, -$) and structured formulas (predicate functions).

This problem was originally presented in a more ambiguous form, using a logic system more powerful than first-order logic.

'*Entscheidungsproblem*' means 'decision problem'

Many tried to solve the problem, without success. One strategy was to try and disprove that such an algorithm can exist. In order to answer this question properly a formal definition of algorithm was required.

1.4 Algorithms

1.4.1 Algorithms Informally

Common features of Algorithms:

Finite	Description of the procedure in terms of elementary operations.
Deterministic	If there is a next step, it is uniquely determined - that is on the same data, the same steps will be made.
Terminate?	Procedure may not terminate on some input data, however we can recognize when it terminates and what the result is.

In 1935/35, Alan Turing (Cambridge) and Church (Princeton) independently gave negative solutions to Hilbert's Entscheidungsproblem (showed such an algorithm could not exist).

1. They gave concrete/precise definitions of what algorithms are (Turing Machines & Lambda Calculus).
2. They regarded algorithms as data, on which other algorithms could act.
3. They reduced the problem to the *Halting problem*.

This work led to the Church-Turing Thesis, that shows everything computable is computed by a Turing Machine. Church's Thesis extended this to show that General Recursive Functions were the same type as those expressed by lambda calculus, and Turing showed that lambda calculus and the Turing machine were equivalent.

Algorithms Formalised

Any formal definition of an algorithm should be:

- Precise** No ambiguities, no implicit assumptions, Should be phrased mathematically.
- Simple** No unnecessary details, only the few axioms required. Makes it easier to reason about.
- General** So all algorithms and types of algorithms are covered.

1.4.2 The Halting Problem

The *Halting problem* is a *decision problem* with:

- The set of all pairs (A, D) such that A is an algorithm, and D is some input datum on which the algorithm operates.
- The property $A(D) \downarrow$ holds for $(A, D) \in S$ if algorithm A when applied to D eventually produces a result (halts).

Turning and Church showed that there is no algorithm such that:

$$\forall (A, D) \in S \left[\begin{array}{ll} H(A, D) & = 1 \quad A(D) \downarrow \\ & 0 \quad \text{otherwise} \end{array} \right]$$

The final step for Turing/Church's proof was to construct an algorithm encoding instances (A, D) of the halting problem as statements such that:

$$\Phi_{A,D} \text{ is provable} \leftrightarrow A(D) \downarrow$$

1.4.3 Algorithms as Functions

It is possible to give a mathematical description of a computable function as a special function between special sets.

In the 1960s Strachey & Scott (Oxford) introduced *denotational semantics*, which describes the meaning (denotation) of an algorithm as a function that maps input to output.

Domains	Definition 1.4.1
<p>Domains are special kinds of partially ordered sets. Partial orders meaning there is an order of elements in the set, but not every element is comparable.</p> <p>Partial orders are reflexive, transitive and anti-symmetric. You can easily represent them on a Hasse Diagram.</p> <div style="text-align: center; margin: 20px 0;"> <pre> graph BT Empty["{}"] --> A["{A}"] Empty --> B["{B}"] Empty --> C["{C}"] A --> AB["{A, B}"] A --> AC["{A, C}"] B --> AB B --> BC["{B, C}"] C --> AC C --> BC AB --> ABC["{A, B, C}"] AC --> ABC BC --> ABC </pre> <p>Diagram of \subseteq for sets $\subseteq \{A, B, C\}$</p> </div>	

Scott solved the most difficult part, considering recursively defined algorithms as continuous functions between domains.

1.4.4 Haskell Programs

Example using a basic implementation of power.

```
-- Precondition: n >= 0
power :: Integer -> Integer -> Integer
power x 0 = 1
```

```

power x n = x * power x (n-1)

-- Precondition: n >= 0
power' :: Integer -> Integer -> Integer
power' x 0 = 1
power' x n
  | even n = k2
  | odd n  = x * k2
where
  k  = power' x (n `div` 2)
  k2 = k * k

O(n)
power 7 5
  ~> 7 * (power 7 4)
  ~> 7 * ( 7 * (power 7 3))
  ~> 7 * ( 7 * (7 * (power 7 2)))
  ~> 7 * ( 7 * (7 * (7 * (power 7 1))))
  ~> 7 * ( 7 * (7 * (7 * (7 * (power 7 0)))))
  ~> 7 * ( 7 * (7 * (7 * (7 * 1))))
  ~> 16807

```

O(log(n)) steps

```

power' 7 5
  ~> 7 * (power' 7 2)2
  ~> 7 * ((power' 7 1)2)2
  ~> 7 * ((7 * (power' 7 0)2)2)2
  ~> 7 * ((7 * (1)2)2)2
  ~> 16807

```

These two functions are equivalent in result however operate differently (one much faster than the other).

1.5 Program Semantics

Denotational Semantics	Definition 1.5.1
<ul style="list-style-type: none"> • A program's meaning is described computationally using denotations (mathematical objects) • A denotation of a program phrase is built from its sub-phrases. 	
Operational Semantics	Definition 1.5.2
Program's meaning is given in terms of the steps taken to make it run.	

There are also *axiomatic semantics* and *declarative semantics* but we will not cover them here.

Chapter 2

While Language

2.1 SimpleExp

We can define a simple expression language (*SimpleExp*) to work on:

$$E \in \text{SimpleExp} ::= n \mid E + E \mid E \times E \mid \dots$$

We want semantics that are the same as we would expect in typical mathematics notation

Small-Set/Structural	Definition 2.1.1	Big-Step/Natural	Definition 2.1.2
Gives a method for evaluating an expression step-by-step.		Ignores intermediate steps and gives result immediately.	
$E \rightarrow E'$		$E \Downarrow n$	

We need big to define big and small step semantics for SimpleExp to describe this, and have those semantics conform to several properties listed.

2.1.1 Big-Step Semantics

Rules

$$\text{(B-NUM)} \frac{}{n \Downarrow n} \quad \text{(B-ADD)} \frac{E_1 \Downarrow n_1 \quad E_2 \Downarrow n_2}{E_1 + E_2 \Downarrow n_3} \quad n_3 = n_1 + n_2$$

We can similarly define multiplication, subtraction etc.

Properties

Determinacy	Definition 2.1.3	Totality	Definition 2.1.4
$\forall E, n_1, n_2. [E \Downarrow n_1 \wedge E \Downarrow n_2 \Rightarrow n_1 = n_2]$		$\forall E. \exists n. [E \Downarrow n]$	
Expression evaluation is deterministic (only one result possible).		Every expression evaluates to something.	

Break it!	Example Question 2.1.1
How could we break the totality of SimpleExp?	
$\text{(B-NON-TOTAL)} \frac{}{true \Downarrow true}$	
We can break <i>totality</i> by introducing a rule that can always match its output.	
The B-NON-TOTAL rule can be applied indefinitely (possible evaluation path that never finishes).	

Show that $3 + (2 + 1) \Downarrow 6$ using the provided rules.

We can hence create the derivation:

$$\frac{\frac{(B-ADD) \frac{(B-NUM) \frac{}{3 \Downarrow 3} (B-ADD) \frac{(B-NUM) \frac{}{2 \Downarrow 2} (B-NUM) \frac{}{1 \Downarrow 1}}{2 + 1 \Downarrow 3}}{3 + (2 + 1) \Downarrow 6}}{}$$

2.1.2 Small Step Semantics

Given a relation \rightarrow we can define its transitive closure \rightarrow^* such that:

$$E \rightarrow^* E' \Leftrightarrow E = E' \vee \exists E_1, E_2, \dots, E_k. [E \rightarrow E_1 \rightarrow E_2 \rightarrow \dots \rightarrow E_k \rightarrow E']$$

Rules

$$\begin{array}{l} \text{(S-ADD)} \frac{}{n_1 + n_2 \rightarrow n_3} \quad n_3 = n_1 + n_2 \\ \text{(S-LEFT)} \frac{E_1 \rightarrow E'_1}{E_1 + E_2 \rightarrow E'_1 + E_2} \quad \text{(S-RIGHT)} \frac{E \rightarrow E'}{n + E \rightarrow n + E'} \end{array}$$

Here we define $+$ as a left-associative operator.

Normal Form

Definition 2.1.5

E is in its normal form (irreducible) if there is no E' such that $E \rightarrow E'$

In *SimpleExp* the normal form is the natural numbers.

Properties

Confluence

Definition 2.1.6

$$\forall E, E_1, E_2. [E \rightarrow^* E_1 \wedge E \rightarrow^* E_2 \Rightarrow \exists E'. [E_1 \rightarrow^* E' \wedge E_2 \rightarrow^* E']]$$

Determinate \rightarrow Confluent

There are several evaluations paths, but they all get the same end result.

Determinacy

Definition 2.1.7

$$\forall E, E_1, E_2. [E \rightarrow E_1 \wedge E \rightarrow E_2 \Rightarrow E_1 = E_2]$$

There is at most one next possible step/rule to apply.

Strong Normalisation

Definition 2.1.8

There are no infinite sequences of expressions, all sequences are finite.

Weak Normalisation

Definition 2.1.9

$$\forall E. \exists k. \exists n. [E \rightarrow^k n]$$

There is some finite sequence of expressions (to normalize) for any expression.

Unique Normal Form

Definition 2.1.10

$$\forall E, n_1, n_2. [E \rightarrow^* n_1 \wedge E \rightarrow n_2 \Rightarrow n_1 = n_2]$$

To be determined...

Example Question 2.1.3

Add a rule to break determinacy without breaking confluence.

$$(\text{S-RIGHT-E}) \frac{E_2 \rightarrow E'_2}{E_1 + E_2 \rightarrow E_1 + E'_2}$$

As we can now choose which side to reduce first (S-LEFT or S-RIGHT-E), we have lost determinacy, however we retain confluence.

2.2 While

2.2.1 Syntax

We can define a simple while language (if, else, while loops) to build programs from & to analyse.

$$\begin{aligned} B \in \text{Bool} &::= \text{true} | \text{false} | E = E | E < E | B \& B | \neg B \dots \\ E \in \text{Exp} &::= x | n | E + E | E \times E | \dots \\ C \in \text{Com} &::= x := E | \text{if } B \text{ then } C \text{ else } C | C; C | \text{skip} | \text{while } B \text{ do } C \end{aligned}$$

Where $x \in \text{Var}$ ranges over variable identifiers, and $n \in \mathbb{N}$ ranges over natural numbers.

2.2.2 States

Partial Function	Definition 2.2.1
A mapping of every member of its domain, to at most one member of its codomain.	

A *state* is a partial function from variables to numbers (partial function as only defined for some variables). For state s , and variable x , $s(x)$ is defined, e.g:

$$s = (x \mapsto 2, y \mapsto 200, z \mapsto 20)$$

(In the current state, $x = 2, y = 200, z = 20$).

For example:

$$\begin{aligned} s[x \mapsto 7](u) &= 7 && \text{if } u = x \\ &= s(u) && \text{otherwise} \end{aligned}$$

The *small-step* semantics of *While* are defined using *configurations* of form:

$$\langle E, s \rangle, \langle B, s \rangle, \langle C, s \rangle$$

(Evaluating E , B , or C with respect to state s)

We can create a new state, where variable x equals value a , from an existing state s :

$$s'(u) \triangleq \alpha(x) = \begin{cases} a & u = x \\ s(u) & \text{otherwise} \end{cases}$$

$$s' = s[x \mapsto u] \text{ is equivalent to } \text{dom}(s') = \text{dom}(s) \wedge \forall y. [y \neq x \rightarrow s(y) = s'(y) \wedge s'(x) = a]$$

(s' equals s where x maps to a)

2.2.3 Rules

Expressions

$$\begin{aligned} (\text{W-EXP.LEFT}) & \frac{\langle E_1, s \rangle \rightarrow_e \langle E'_1, s' \rangle}{\langle E_1 + E_2, s \rangle \rightarrow_e \langle E'_1 + E_2, s' \rangle} & (\text{W-EXP.RIGHT}) & \frac{\langle E, s \rangle \rightarrow_e \langle E', s' \rangle}{\langle n + E, s \rangle \rightarrow_e \langle n + E', s' \rangle} \\ (\text{W-EXP.VAR}) & \frac{}{\langle x, s \rangle \rightarrow_e \langle n, s \rangle} \quad s(x) = n & (\text{W-EXP.ADD}) & \frac{}{\langle n_1 + n_2, s \rangle} \quad \langle n_3, s \rangle n_3 = n_1 + n_2 \end{aligned}$$

These rules allow for side effects, despite the While language being side effect free in expression evaluation. We show this by changing state $s \rightarrow_e s'$.

We can show inductively (from the base cases W-EXP.VAR and W-EXP.ADD) that expression evaluation is side effect free.

Booleans

(Based on expressions, one can create the same for booleans) ($b \in \{\text{true}, \text{false}\}$)

$$\begin{aligned} (\text{W-BOOL.AND.LEFT}) & \frac{\langle B_1, s \rangle \rightarrow_b \langle B'_1, s' \rangle}{\langle B_1 \& B_2, s \rangle \rightarrow_b \langle B'_1 \& B_2, s' \rangle} & (\text{W-BOOL.AND.RIGHT}) & \frac{\langle B, s \rangle \rightarrow_b \langle B', s' \rangle}{\langle b \& B_2, s \rangle \rightarrow_b \langle b \& B', s' \rangle} \\ (\text{W-BOOL.AND.TRUE}) & \frac{}{\langle \text{true} \& \text{true}, s \rangle \rightarrow_b \langle \text{true}, s \rangle} & (\text{W-BOOL.AND.FALSE}) & \frac{}{\langle \text{false} \& b, s \rangle \rightarrow_b \langle \text{true}, s \rangle} \end{aligned}$$

(Notice we do not short circuit, as the right arm may change the state. In a side effect free language, we could.)

$$\begin{array}{ll}
(\text{W-BOOL.EQUAL.LEFT}) \frac{\langle E_1, s \rangle \rightarrow_e \langle E'_1, s' \rangle}{\langle E_1 = E_2, s \rangle \rightarrow_b \langle E'_1 = E_2, s' \rangle} & (\text{W-BOOL.EQUAL.RIGHT}) \frac{\langle E, s \rangle \rightarrow_e \langle E', s' \rangle}{\langle n = E, s \rangle \rightarrow_b \langle n = E, s' \rangle} \\
(\text{W-BOOL.EQUAL.TRUE}) \frac{}{\langle n_1 = n_2, s \rangle \rightarrow_b \langle \text{true}, s \rangle} n_1 = n_2 & (\text{W-BOOL.EQUAL.FALSE}) \frac{}{\langle n_1 = n_2, s \rangle \rightarrow_b \langle \text{false}, s \rangle} n_1 \neq n_2 \\
(\text{W-BOOL.LESS.LEFT}) \frac{\langle E_1, s \rangle \rightarrow_e \langle E'_1, s' \rangle}{\langle E_1 < E_2, s \rangle \rightarrow_b \langle E'_1 < E_2, s' \rangle} & (\text{W-BOOL.LESS.RIGHT}) \frac{\langle E, s \rangle \rightarrow_e \langle E', s' \rangle}{\langle n < E, s \rangle \rightarrow_b \langle n < E, s' \rangle} \\
(\text{W-BOOL.LESS.TRUE}) \frac{}{\langle n_1 < n_2, s \rangle \rightarrow_b \langle \text{true}, s \rangle} n_1 < n_2 & (\text{W-BOOL.EQUAL.FALSE}) \frac{}{\langle n_1 < n_2, s \rangle \rightarrow_b \langle \text{false}, s \rangle} n_1 \geq n_2 \\
(\text{W-BOOL.NOT}) \frac{}{\langle \neg \text{true}, s \rangle \rightarrow_b \langle \text{false}, s \rangle} & (\text{W-BOOL.NOT}) \frac{}{\langle \neg \text{false}, s \rangle \rightarrow_b \langle \text{true}, s \rangle}
\end{array}$$

Assignment

$$\begin{array}{ll}
(\text{W-ASS.EXP}) \frac{\langle E, s \rangle \rightarrow_e \langle E', s' \rangle}{\langle x := E, s \rangle \rightarrow_c \langle x := E', s' \rangle} & (\text{W-ASS.NUM}) \frac{}{\langle x := n, s \rangle \rightarrow_c \langle \text{skip}, s[x \mapsto n] \rangle}
\end{array}$$

Sequential Composition

$$\begin{array}{ll}
(\text{W-SEQ.LEFT}) \frac{\langle C_1, s \rangle \rightarrow_c \langle C'_1, s' \rangle}{\langle C_1; C_2, s \rangle \rightarrow_c \langle C'_1; C_2, s' \rangle} & (\text{W-SEQ.SKIP}) \frac{}{\langle \text{skip}; C, s \rangle \rightarrow_c \langle C, s \rangle}
\end{array}$$

Conditionals

$$\begin{array}{ll}
(\text{W-COND.TRUE}) \frac{}{\langle \text{if true then } C_1 \text{ else } C_2, s \rangle \rightarrow_c \langle C_1, s \rangle} & \\
(\text{W-COND.FALSE}) \frac{}{\langle \text{if false then } C_1 \text{ else } C_2, s \rangle \rightarrow_c \langle C_2, s \rangle} & \\
(\text{W-COND.BEXP}) \frac{\langle B, s \rangle \rightarrow_b \langle B', s' \rangle}{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, s \rangle \rightarrow_c \langle \text{if } B' \text{ then } C_1 \text{ else } C_2, s' \rangle} &
\end{array}$$

While

$$(\text{W-WHILE}) \frac{}{\langle \text{while } B \text{ do } C, s \rangle \rightarrow_c \langle \text{if } B \text{ then } (C; \text{while } B \text{ do } C) \text{ else skip}, s \rangle}$$

2.2.4 Properties

The execution relation (\rightarrow_c) is deterministic.

$$\forall C, C_1, C_2 \in \text{Com} \forall s, s_1, s_2. [\langle C, s \rangle \rightarrow_c \langle C_1, s_1 \rangle \wedge \langle C, s \rangle \rightarrow_c \langle C_2, s_2 \rangle \rightarrow \langle C_1, s_1 \rangle = \langle C_2, s_2 \rangle]$$

Hence the relation is also confluent:

$$\begin{array}{l}
\forall C, C_1, C_2 \in \text{Com} \forall s, s_1, s_2. [\langle C, s \rangle \rightarrow_c \langle C_1, s_1 \rangle \wedge \langle C, s \rangle \rightarrow_c \langle C_2, s_2 \rangle \rightarrow \\
\exists C' \in \text{Com}, s'. [\langle C_1, s_1 \rangle \rightarrow_c \langle C', s' \rangle \wedge \langle C_2, s_2 \rangle \rightarrow_c \langle C', s' \rangle]]
\end{array}$$

Both also hold for \rightarrow_e and \rightarrow_b .

2.2.5 Configurations

Answer Configuration

A configuration $\langle \text{skip}, s \rangle$ is an *answer configuration*. As there is no rule to execute skip, it is a normal form.

$$\neg \exists C \in \text{Com}, s, s'. [\langle \text{skip}, s \rangle \rightarrow_c \langle C, s' \rangle]$$

For booleans $\langle \text{true}, s \rangle$ and $\langle \text{false}, s \rangle$ are answer configurations, and for expressions $\langle n, s \rangle$.

Stuck Configurations

A configuration that cannot be evaluated to a normal form is called a *stuck configuration*.

$$\langle y, (x \mapsto 3) \rangle$$

Note that a configuration that leads to a *stuck configuration* is not itself stuck.

$$\langle 5 < y, (x \mapsto 2) \rangle$$

(Not stuck, but reduces to a stuck state)

2.2.6 Normalising

The relations \rightarrow_b and \rightarrow_e are normalising, but \rightarrow_c is not as it may not have a normal form.

while *true* do *skip*

$$\langle \text{while } \textit{true} \text{ do } \textit{skip}, s \rangle \rightarrow_c^3 \langle \text{while } \textit{true} \text{ do } \textit{skip}, s \rangle$$

(\rightarrow_c^3 means 3 steps, as we have gone through more than one to get the same configuration, it is an infinite loop)

2.2.7 Side Effecting Expressions

If we allow programs such as:

do $x := x + 1$ return x

$$(\text{do } x := x + 1 \text{ return } x) + (\text{do } x := x \times 1 \text{ return } x)$$

(value depends on evaluation order)

2.2.8 Short Circuit Semantics

$$\frac{B_1 \rightarrow_b B'_1}{B_1 \& B_2 \rightarrow_b B'_1 \& B_2} \quad \frac{}{false \& B \rightarrow_b false} \quad \frac{}{true \& B \rightarrow_b B}$$

2.2.9 Strictness

An operation is *strict* when arguments must be evaluated before the operation is evaluated. Addition is strict as both expressions must be evaluated (left, then right).

Due to short circuiting, $\&$ is left strict as it is possible for the operation to be evaluated without evaluating the right (*non-strict* in right argument).

2.2.10 Complex Programs

It is now possible to build complex programs to be evaluated with our small step rules.

$$Factorial \triangleq y := x; a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$$

We can evaluate *Factorial* with an input $s = [x \mapsto \dots]$ to get answer configuration $[\dots, a \mapsto x!, x \mapsto \dots]$

Execute!	Example Question 2.2.1
<p>Evaluate <i>Factorial</i> for the following initial configuration:</p> <p style="text-align: center;">$s = [x \mapsto 3, y \mapsto 17, z \mapsto 42]$</p> <hr style="border-top: 1px dashed #8B4513;"/> <p>Start</p> <p style="text-align: center;">$\langle y := x; a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), [x \mapsto 3, y \mapsto 17, z \mapsto 42] \rangle$</p> <p>Get x variable</p> <p>where $C = a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 17, z \mapsto 42)$:</p> $(W-SEQ.LEFT) \frac{(W-ASS.EXP) \frac{(W-EXP.VAR) \frac{\langle x, s \rangle \rightarrow_e \langle 3, s \rangle}{\langle y := x, s \rangle \rightarrow_c \langle y := 3, s \rangle}}{\langle y := x; C, s \rangle \rightarrow_c \langle y := 3; C, s \rangle}}{\langle y := x; C, s \rangle \rightarrow_c \langle y := 3; C, s \rangle}}$ <p>Result:</p> <p style="text-align: center;">$\langle y := 3; a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 17, z \mapsto 42) \rangle$</p> <p>Assign to y variable</p> <p>where $C = a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 17, z \mapsto 42)$:</p> $(W-SEQ.LEFT) \frac{(W-ASS.NUM) \frac{\langle y := 3, s \rangle \rightarrow_c \langle \text{skip}, s[y \mapsto 3] \rangle}{\langle y := 3; C, s \rangle \rightarrow_c \langle \text{skip}; C, s[y \mapsto 3] \rangle}}{\langle y := 3; C, s \rangle \rightarrow_c \langle \text{skip}; C, s[y \mapsto 3] \rangle}}$ <p>Result:</p> <p style="text-align: center;">$\langle \text{skip}; a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42) \rangle$</p>	

Eliminate skip

where $C = a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1) \text{ and } s = (x \mapsto 3, y \mapsto 3, z \mapsto 42)$:

$$(W\text{-SEQ.SKIP}) \frac{}{\langle \text{skip}; C, s \rangle \rightarrow_c \langle C, s \rangle}$$

Result:

$$\langle a := 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42) \rangle$$

Assign a

where $C = \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1) \text{ and } s = (x \mapsto 3, y \mapsto 3, z \mapsto 42)$:

$$(W\text{-SEQ.LEFT}) \frac{(W\text{-ASS.NUM}) \frac{}{\langle a := 1, s \rangle \rightarrow_c \langle \text{skip}, s[a \mapsto 1] \rangle}}{\langle a := 1; C, s \rangle \rightarrow_c \langle \text{skip}; C, s[a \mapsto 1] \rangle}$$

Result:

$$\langle \text{skip}; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Eliminate skip

where $C = \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1) \text{ and } s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$

$$(W\text{-SEQ.SKIP}) \frac{}{\langle \text{skip}; C, s \rangle \rightarrow_c \langle C, s \rangle}$$

Result:

$$\langle \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Expand while

where $C = (a := a \times y; y := y - 1)$, $B = 0 < y$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$(W\text{-WHILE}) \frac{}{\langle \text{while } B \text{ do } C, s \rangle \rightarrow_c \langle \text{if } B \text{ then } (C; \text{while } B \text{ do } C) \text{ else skip}, s \rangle}$$

Result:

$$\langle \text{if } 0 < y \text{ then } (a := a \times y; y := y - 1; \text{while } 0 < y \text{ do } a := a \times y; y := y - 1) \text{ else skip}, (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Get y variable

where $C = (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$(W\text{-COND.BEXP}) \frac{(W\text{-BOOL.LESS.RIGHT}) \frac{(W\text{-EXP.VAR}) \frac{}{\langle y, s \rangle \rightarrow \langle 3, s \rangle}}{\langle 0 < y, s \rangle \rightarrow_b \langle 0 < 3, s \rangle}}{\langle \text{if } 0 < y \text{ then } (C; \text{while } 0 < y \text{ do } C) \text{ else skip}, s \rangle \rightarrow_c \langle \text{if } 0 < 3 \text{ then } (C; \text{while } 0 < y \text{ do } C) \text{ else skip}, s \rangle}$$

Result:

$$\langle \text{if } 0 < 3 \text{ then } (a := a \times y; y := y - 1; \text{while } 0 < y \text{ do } a := a \times y; y := y - 1); \text{ else skip}, (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Complete if boolean

where $C = (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$(W\text{-COND.EXP}) \frac{(W\text{-BOOL.LESS.TRUE}) \frac{}{\langle 0 < 3, s \rangle \rightarrow_b \langle \text{true}, s \rangle}}{\langle \text{if } 0 < 3 \text{ then } (C; \text{while } 0 < y \text{ do } C) \text{ else skip}, s \rangle \rightarrow_c \langle \text{if true then } (C; \text{while } 0 < y \text{ do } C) \text{ else skip}, s \rangle}$$

Result:

$$\langle \text{if true then } (a := a \times y; y := y - 1; \text{while } 0 < y \text{ do } a := a \times y; y := y - 1); \text{ else skip}, (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Evaluate if

where $C = (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$(W\text{-COND.TRUE}) \frac{}{\langle \text{if true then } (C; \text{while } 0 < y \text{ do } C) \text{ else skip}, s \rangle \rightarrow_c \langle C; \text{while } 0 < y \text{ do } C, s \rangle}$$

Result:

$$\langle a := a \times y; y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Evaluate Expression a

where $C = y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$\begin{array}{c} \text{(W-EXP.VAR)} \frac{}{\langle a, s \rangle \rightarrow \langle 1, s \rangle} \\ \text{(W-EXP.MUL.LEFT)} \frac{}{\langle a \times y, s \rangle \rightarrow_e \langle 1 \times y, s \rangle} \\ \text{(W-ASS.EXP)} \frac{}{\langle a := a \times y, s \rangle \rightarrow_c \langle a := 1 \times y, s \rangle} \\ \text{(W-SEQ.LEFT)} \frac{}{\langle a := a \times y; C, s \rangle \rightarrow_c \langle a := 1 \times y; C, s \rangle} \end{array}$$

Result:

$$\langle a := 1 \times y; y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Evaluate Expression y

where $C = y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$\begin{array}{c} \text{(W-EXP.VAR)} \frac{}{\langle y, s \rangle \rightarrow_e \langle 3, s \rangle} \\ \text{(W-EXP.MUL.RIGHT)} \frac{}{\langle 1 \times y, s \rangle \rightarrow_e \langle 1 \times 3, s \rangle} \\ \text{(W-ASS.EXP)} \frac{}{\langle a := 1 \times y, s \rangle \rightarrow_c \langle a := 1 \times 3, s \rangle} \\ \text{(W-SEQ.LEFT)} \frac{}{\langle a := 1 \times y; C, s \rangle \rightarrow \langle a := 1 \times 3; C, s \rangle} \end{array}$$

Result:

$$\langle a := 1 \times 3; y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Evaluate Multiply

where $C = y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$\begin{array}{c} \text{(W-EXP.MUL)} \frac{}{\langle 1 \times 3, s \rangle \rightarrow_e \langle 3, s \rangle} \\ \text{(W-ASS.EXP)} \frac{}{\langle a := 1 \times 3, s \rangle \rightarrow_c \langle a := 3, s \rangle} \\ \text{(W-SEQ.LEFT)} \frac{}{\langle a := 1 \times 3; C, s \rangle \rightarrow_c \langle a := 3; C, s \rangle} \end{array}$$

Result:

$$\langle a := 3; y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1) \rangle$$

Assign 3 to a

where $C = y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 1)$:

$$\begin{array}{c} \text{(W-ASS.NUM)} \frac{}{\langle a := 3, s \rangle \rightarrow_c \langle \text{skip}, s[a \mapsto 3] \rangle} \\ \text{(W-SEQ.LEFT)} \frac{}{\langle a := 3; C, s \rangle \rightarrow_c \langle \text{skip}; C, s[a \mapsto 3] \rangle} \end{array}$$

Result:

$$\langle \text{skip}; y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3) \rangle$$

Eliminate Skip

where $C = y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3)$:

$$\text{(W-SEQ.SKIP)} \frac{}{\langle \text{skip}; C, s \rangle \rightarrow_c \langle C, s \rangle}$$

Result:

$$\langle y := y - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3) \rangle$$

Assign 3 to y

where $C = \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1)$ and $s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3)$:

$$\begin{array}{c} \text{(W-EXP.VAR)} \frac{}{\langle y, s \rangle \rightarrow \langle 3, s \rangle} \\ \text{(W-EXP.SUB.LEFT)} \frac{}{\langle y - 1, s \rangle \rightarrow_e \langle 3 - 1, s \rangle} \\ \text{(W-ASS.EXP)} \frac{}{\langle y := y - 1, s \rangle \rightarrow_c \langle y := 3 - 1, s \rangle} \\ \text{(W-SEQ.LEFT)} \frac{}{\langle y := y - 1; C, s \rangle \rightarrow_c \langle y := 3 - 1, s \rangle} \end{array}$$

Result:

$$\langle y := 3 - 1; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3) \rangle$$

Evaluate Subtraction

where $C = \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1) \text{ and } s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3)$:

$$\begin{array}{c} \text{(W-EXP.SUB)} \frac{}{\langle 3 - 1, s \rangle \rightarrow_e \langle 2, s \rangle} \\ \text{(W-ASS.EXP)} \frac{}{\langle y := 3 - 1, s \rangle \rightarrow_c \langle y := 2, s \rangle} \\ \text{(W-SEQ.LEFT)} \frac{}{\langle y := 3 - 1; C, s \rangle \rightarrow_c \langle y := 2; C, s \rangle} \end{array}$$

Result:

$$\langle y := 2; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3) \rangle$$

Assign 2 to y

where $C = \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1) \text{ and } s = (x \mapsto 3, y \mapsto 3, z \mapsto 42, a \mapsto 3)$:

$$\begin{array}{c} \text{(W-ASS.NUM)} \frac{}{\langle y := 2, s \rangle \rightarrow_c \langle \text{skip}, s[y \mapsto 2] \rangle} \\ \text{(W-SEQ.LEFT)} \frac{}{\langle y := 2; C, s \rangle \rightarrow_c \langle \text{skip}; C, s[y \mapsto 2] \rangle} \end{array}$$

Result:

$$\langle \text{skip}; \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 2, z \mapsto 42, a \mapsto 3) \rangle$$

Eliminate skip

where $C = \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1) \text{ and } s = (x \mapsto 3, y \mapsto 2, z \mapsto 42, a \mapsto 3)$:

$$\text{(W-SEQ.SKIP)} \frac{}{\langle \text{skip}; C, s \rangle \rightarrow_c \langle C, s \rangle}$$

Result:

$$\langle \text{while } 0 < y \text{ do } (a := a \times y; y := y - 1), (x \mapsto 3, y \mapsto 2, z \mapsto 42, a \mapsto 3) \rangle$$

UNFINISHED!!!

Chapter 3

Credit

Image Credit

Front Cover Analytical Engine - Science Museum London

Content

Based on the *Models of Computation* course taught by Dr Azelea Raad and Dr Herbert Wiklicky.

These notes were written by Oliver Killane.