50003 - Models of Computation - Lecture 1

Oliver Killane

14/10/21

Lecture Recording

Lecture recording is available here

Course Admin

Part 1 - Azalea Raad (Week 2 - Week 6)

- **8 Virtual Lectures - Teams & Recorded** Monday (11:00 - 12:00) Friday (16:00 - 17:00)
- **4 Hybrid tutorials - Teams & In Person** Friday (17:00 - 18:00)
- **1 Bonus Hour - Teams & Recorded** Monday 8/11/2012

We will cover:

- Operational Semantics of a small while language
- Denotational semantics of a small while language (notes only)
- Register machines, universal register machine, Halting problem.
- Turing machines and Turing computable functions, primitive and partial recursive functions
- Lambda calculus and equivalence results.

Part 2 - Herbert Wiklicky (Week 6 - Week 10)

- **Hybrid Lectures - Teams & In Person & Recorded** Friday (16:00 - 18:00)
- **Virtual Tutorials - Teams** Monday (11:00 - 12:00)

Algorithms

Examples of Algorithms

- **Euclid's Algorithm ≈300 B.C.** Algorithm to find the greatest common divisior.

```haskell
-- Euclid's algorithm:
-- 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 ≈200 B.C.** Used to find the prime numbers within a limit. Done by starting from the 2, adding the number to the primes, then marking all multiples, then repeating progressing to the next non-marked number (a prime), marking all multiples and repeating.

```haskell
-- Sieve of Eratosthenes
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 [] = []
```

- **Well-Known Rules for $+ - \div \times$ ≈900 A.D.** Al-Khwarizmi, a persian mathematician who was appointed astronomer and head of the library in the Bagdad House of Wisdom.
- **Simple Machines using punchcards. Mainly 19th Century** Weaving looms, pianola, census tabulating machine.
- **Analytical Engine** A proposed multi-purpose calculating machine designed by Charles Babbage. Using a simple ALU, with conditional branching and integrated memory. As a result the first Turing complete computer.

  First ever program was written by Ada Lovelace to calculate the Bernoulli numbers.

Decision Problems

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 porperty of a formula that it has a proof).

Formulas

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

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.

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

Also called predicate logic, is an extension of propositional logic that includes quanifiers ($\forall, \exists$), 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.

'*Entscheidungsproblem*' means 'decision problem'

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

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.
- **Maybe Terminate** procedure may not terminate on some input data, however we can recognise when it terminates and what the result is.

In 1935/35, Alan Turing (Cambridge) and Church (Princeton) independently gave negative soltuions to Hilberts 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 Recurisve Functions were the same type as those expressed by lambda calculus, and Turning showed that lambda calculus and the turning 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.

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:

$$\forall (A, D) \in S \begin{bmatrix} H(A,D) & = & 1 & A(D)\downarrow \\ & & 0 & otherwise \end{bmatrix}$$

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:

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

Algorithms as Functions

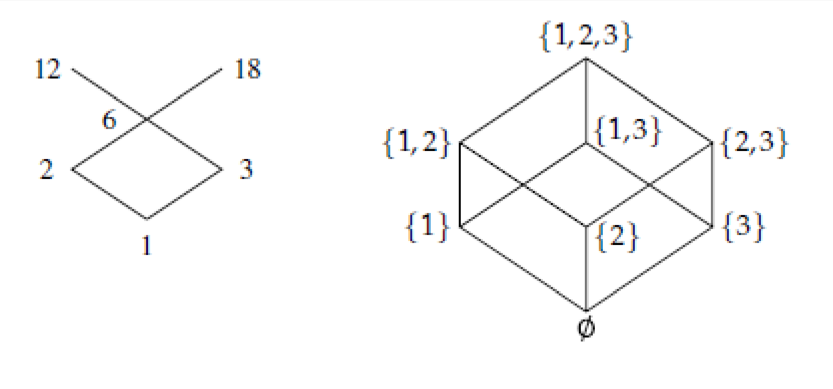
It is possible to give a mathematical description of a computable function as a special function between special sets.

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

Domains

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.

Partial orders are reflexive, transitive and anti-symmetric. You can easily represent them on a Hasse Diagram.



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

Haskell Programs

Example using a basic implementation of power.

```haskell
-- Precondition: n >= 0
power :: Integer -> Integer -> Integer
power x 0 = 1
power x n = x * power x (n-1)

-- Precondition: n >= 0
```

```haskell
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)2̂
⇝ 7 * ((power' 7 1)2̂)2̂
⇝ 7 * ((7 * (power' 7 0)2̂)2̂)2̂
⇝ 7 * ((7 * (1)2̂)2̂)2̂
⇝ 16807

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

Program Semantics

Denotational Semantics:

- A program's meaning is described compositionally using denotations (mathematical objects)
- A denotation of a program phrase is built from its subphrases.

Operational Semantics:

- Program's meaning is given in terms of the steps taken to make it run.

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