Computing with functions.
Partial orders.

Computation?

How much do we need to perform some computation?
For example, some algorithm from your C++ homework assignment.

- conditional branching
- loops
- good to have some data structures
- variables and code abstraction (objects, functions)

Can we do it by using only functions?

For convenience we assume that we have natural numbers and the operator $+$ for adding them.

Simple functions

Constant function

$$x \mapsto 23$$

Identity function

$$x \mapsto x$$

Successor function

$$x \mapsto x + 1$$

Function application

If f is a function, the usual notation

$$f(x)$$

denotes a function application to the argument x.

We are going to use a shorter notation for application:

$$f \ \ x$$

Application is left-associative (just like $+$, $-$, $\times$):

$$f \ \ x \ \ y \equiv (f \ \ x) \ \ y$$

$$f \ \ x \ \ y \ \ z \equiv ((f \ \ x) \ \ y) \ \ z$$

Function application

Applying our functions to the argument 7:

$$(x \mapsto 23)\ 7 \implies 23$$
$$(x \mapsto x)\ 7 \implies 7$$
$$(x \mapsto x+1)\ 7 \implies 7+1 \implies 8$$

This is really boring! The computations are trivial.

Functions that return functions

Consider a function that takes an argument x and returns another function that always returns x

$$x \mapsto (y \mapsto x)$$

Applying it to the argument 5:

$$(x \mapsto (y \mapsto x))\ 5 \quad \Longrightarrow \quad y \mapsto 5$$

So the result of the application is a constant function $y \mapsto 5$.

Functions that return functions

Another example:

$$x \mapsto (y \mapsto y)$$

Applying it to the argument 12:

$$(x \mapsto (y \mapsto y))\ 12 \quad \Longrightarrow \quad y \mapsto y$$

It drops the argument and returns an identity function.

Two or more arguments

Addition function (using operator + internally):

$$x \mapsto (y \mapsto x + y)$$

Applying it to the arguments 5 and 7:

$$\begin{aligned} (x \mapsto (y \mapsto x + y))\ 5\ 7 &\implies (y \mapsto 5 + y)\ 7 \\ &\implies 5 + 7 \\ &\implies 12 \end{aligned}$$

It also resembles sequential composition and variable binding:

$x = 5;$
$y = 7;$
return $x + y;$

Observation

Consider two previously mentioned functions:

$$x \mapsto (y \mapsto x)\ 1\ 2 \quad \Longrightarrow \quad 1$$

$$x \mapsto (y \mapsto y)\ 1\ 2 \quad \Longrightarrow \quad 2$$

Can we use this behavior for doing something useful?

Ifthenelse

Function *Ifthenelse*:

$$c \mapsto (a \mapsto (b \mapsto c \;\; a \;\; b))$$

Function *True*:

$$x \mapsto (y \mapsto x)$$

Function *False*:

$$x \mapsto (y \mapsto y)$$

$$\begin{aligned}
&\mathit{Ifthenelse}\;\; \mathit{True}\;\; 1\;\; 2 \\
&\quad \Longrightarrow \quad \big(c \mapsto (a \mapsto (b \mapsto c \;\; a \;\; b))\big)\;\; \mathit{True}\;\; 1\;\; 2 \\
&\quad \Longrightarrow \quad (a \mapsto (b \mapsto \mathit{True}\;\; a\;\; b))\;\; 1\;\; 2 \\
&\quad \Longrightarrow \quad (b \mapsto \mathit{True}\;\; 1\;\; b)\;\; 2 \\
&\quad \Longrightarrow \quad \mathit{True}\;\; 1\;\; 2 \\
&\quad \Longrightarrow \quad (x \mapsto (y \mapsto x))\;\; 1\;\; 2 \quad \Longrightarrow \quad (y \mapsto 1)\;\; 2 \quad \Longrightarrow \quad 1
\end{aligned}$$

Ordered Pair

How to construct ordered pairs?

We need to implement three function:

- Pair construction

$$\text{“}\mathit{Pair}\ \ a\ \ b = (a, b)\text{”}$$

- Projection function that returns the first element

$$\text{“}\mathit{First}\ \ (a, b) = a\text{”}$$

- Projection function that returns the second element

$$\text{“}\mathit{Second}\ \ (a, b) = b\text{”}$$

Ordered Pair

Function *Pair*:

$$a \mapsto (b \mapsto (c \mapsto c \;\; a \;\; b))$$

Function *First*:

$$x \mapsto x \;\; \mathit{True}$$

Function *Second*:

$$x \mapsto x \;\; \mathit{False}$$

$$\begin{aligned}
&\mathit{Second} \;\; (\mathit{Pair} \;\; 5 \;\; 7) \\
&\quad\Longrightarrow \quad (x \mapsto x \;\; \mathit{False}) \;\; (\mathit{Pair} \;\; 5 \;\; 7) \\
&\quad\Longrightarrow \quad (\mathit{Pair} \;\; 5 \;\; 7) \;\; \mathit{False} \\
&\quad\Longrightarrow \quad \big(a \mapsto (b \mapsto (c \mapsto c \;\; a \;\; b))\big) \;\; 5 \;\; 7 \;\; \mathit{False} \\
&\quad\Longrightarrow \quad (b \mapsto (c \mapsto c \;\; 5 \;\; b)) \;\; 7 \;\; \mathit{False} \\
&\quad\Longrightarrow \quad (c \mapsto c \;\; 5 \;\; 7) \;\; \mathit{False} \\
&\quad\Longrightarrow \quad \mathit{False} \;\; 5 \;\; 7 \quad \Longrightarrow \quad \cdots \quad \Longrightarrow \quad 7
\end{aligned}$$

Function M

Function M:

$$x \mapsto x \;\; x$$

It returns its argument applied to itself.

Let's apply this function to something. Any suggestions?

$$\begin{aligned}(x \mapsto x \;\; x) \; (y \mapsto y) &\implies (y \mapsto y) \; (y \mapsto y) \\ &\implies y \mapsto y\end{aligned}$$

Better suggestions?

Function M

Function M:

$$x \mapsto x \;\; x$$

Apply it to itself:

$$(x \mapsto x \;\; x) \;\; (x \mapsto x \;\; x) \quad \Longrightarrow$$

Function M

Function M:

$$x \mapsto x \;\; x$$

Apply it to itself:

$$\begin{aligned} (x \mapsto x \;\; x) \; (x \mapsto x \;\; x) &\Longrightarrow (x \mapsto x \;\; x) \; (x \mapsto x \;\; x) \\ &\Longrightarrow (x \mapsto x \;\; x) \; (x \mapsto x \;\; x) \\ &\Longrightarrow \ldots \end{aligned}$$

This is an infinite loop, something like

while(true) { ; }

Based on this principle, we can implement real recursion and loops that actually do something.

Lambda calculus

This computational formalism is called lambda calculus. This is a universal model of computation, in the sense that your laptop cannot compute anything what cannot be computed in lambda calculus.

It was introduced by Alonzo Church in 1930s.

We need to fix the notation.

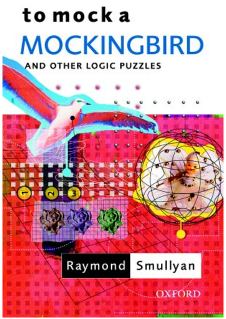
$$\text{Instead of} \quad (x \mapsto x \ \ y \ \ z),$$

$$\text{we write} \quad \lambda x\,.\,xyz$$

Also, you need to be careful with the names of the variables, to make substitutions correctly.

The order of evaluation (which function application gets reduced first?) is important, and it has to be defined precisely.

Further reading on the topic



"To Mock a Mockingbird and Other Logic Puzzles" (chapter 3)

by Raymond Smullyan

$$Mx \implies xx$$

Also, you can try learning functional programming languages like Scheme, Erlang, ML, or Haskell.

Almost every modern programming language (say, developed after 2000) has some functional features. Even JavaScript has Scheme-like functional core.

Relations

Remember that a relation is a subset of the Cartesian Product of two sets.

For example,

$$R = \{(a, b) \in A \times B \mid \text{some property holds}\}$$

$$R \subseteq A \times B$$

For convenience, we adopt the following infix notation:

$$\text{when} \quad (a, b) \in R, \quad \text{we write} \quad aRb$$

Relations. Infix notation

It is originated from the relations like $=$, $\leq$, $\geq$, $<$, and $>$.

$$(1,2) \in R_{(<)} \quad \text{we usually write} \quad 1 < 2$$

$$(3,3) \in R_{(=)} \quad \text{we usually write} \quad 3 = 3$$

Divisibility is a relation on $\mathbb{N}$ too. And we use infix notation:

$$(15,60) \in R_{(divides)} \quad \text{we write} \quad 15 \mid 60$$

Relations on the same set

What if the sets A and B are the same?

$$R \subseteq A \times A$$

For example, $=$, $\leq$, $\geq$, $<$, $>$ are relations on $\mathbb{N}$. That is, these relations are subsets of $\mathbb{N} \times \mathbb{N}$.

Def. A relation on the set A is

- *reflexive* if $\forall x \in A: xRx$.
- *symmetric* if $\forall x, y \in A: xRy \to yRx$.
- *antisymmetric* if $\forall x, y \in A: (xRy \land yRx) \to x = y$.
- *transitive* if $\forall x, y, z \in A: (xRy \land yRz) \to xRz$.

Relations on the same set

- *reflexive* if $\forall x \in A : xRx$.
- *symmetric* if $\forall x, y \in A : xRy \rightarrow yRx$.
- *antisymmetric* if $\forall x, y \in A : (xRy \land yRx) \rightarrow x = y$.
- *transitive* if $\forall x, y, z \in A : (xRy \land yRz) \rightarrow xRz$.

	reflexive?	symmetric?	antisymmetric?	transitive?
$x \equiv y \pmod 5$	Yes	Yes	No	Yes
$x \mid y$	Yes	No	Yes	Yes
$x \leq y$	Yes	No	Yes	Yes

Partial orders

Def. A relation is a *partial order* if it is reflexive, antisymmetric, and transitive.

An example, the "divides" relation on the natural numbers is a partial order:

- It is reflexive because $x \mid x$.
- It is antisymmetric because $x \mid y$ and $y \mid x$ implies $x = y$.
- It is transitive because $x \mid y$ and $y \mid z$ implies $x \mid z$.

The $\leq$ relation on the natural numbers is also a partial order. However, the $<$ relation is not a partial order, because it is not reflexive; no number is less than itelf.

Partial orders

Often a partial order relation is denoted with the symbol

$$\preceq$$

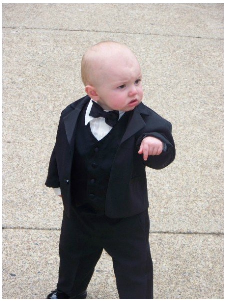
instead of a letter, like R.

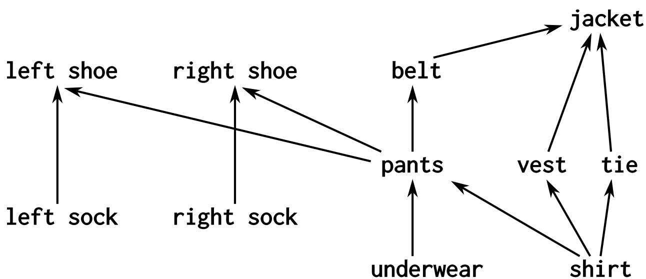
This makes sense since the symbol calls to mind $\leq$, which is one of the most common partial orders.

$x \preceq y$ it reads as “x precedes y”.

Partially ordered sets

Def. If $\preceq$ is a partial order on the set A, then the pair $(A, \preceq)$ is called a *partially-ordered set* or *poset*.

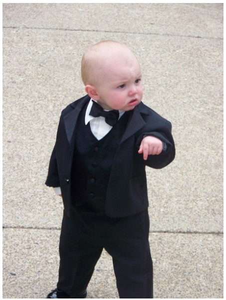


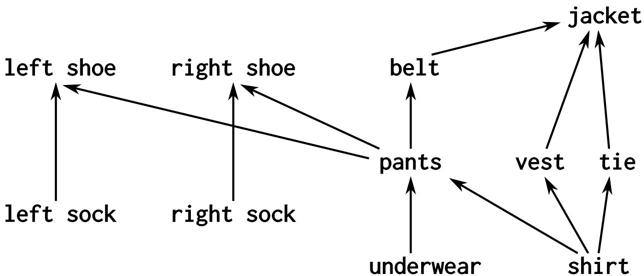


Def. The elements x and y of a poset $(A, \preceq)$ are called *comparable* if either $x \preceq y$ or $x \preceq y$.

When x and y are elements of A such that neither $x \preceq y$ nor $y \preceq x$, x and y are called *incomparable*.

Hasse diagram





This graph is called the *Hasse diagram* for the poset $(A, \preceq)$.

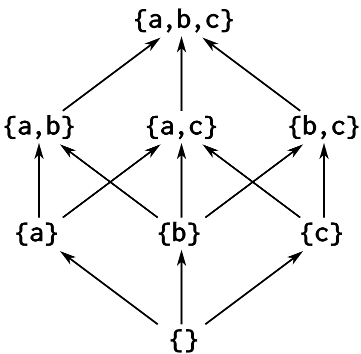
For a and b from A, we draw an edge from a to b if $a \preceq b$.

Self-loops and edges implied by transitivity are omitted.

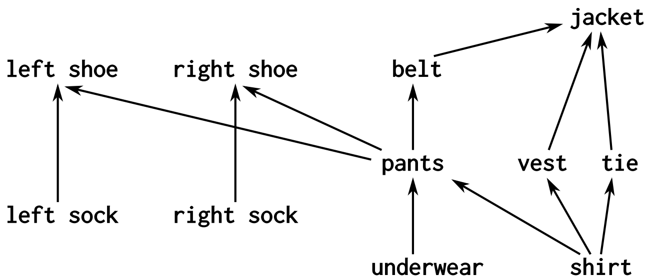
Hasse diagram

Consider a poset $(\mathscr{P}(A), \subseteq)$ for $A = \{a, b, c\}$.

Its Hasse diagram:



Maximal and minimal elements



An element of a poset is called *maximal* if it is not less than any other element of the poset. $a \in A$ is maximal in the poset $(A, \preceq)$ if there is no $b \in A$ such that $a \prec b$.

Similarly, an element of a poset is called *minimal* if it is not greater than any other element of the poset. That is, a is minimal if there is no element $b \in A$ such that $b \prec a$.

Partially ordered sets

Theorem. A poset $(A, \preceq)$ has no directed cycles other than self-loops, that is, there is no sequence of $n \geq 2$ distinct elements $a_i \in A$ such that

$$a_1 \preceq a_2 \preceq a_3 \preceq a_4 \preceq \ldots \preceq a_{n-1} \preceq a_n \preceq a_1$$

Partially ordered sets

Theorem. A poset $(A, \preceq)$ has no directed cycles other than self-loops, that is, there is no sequence of $n \geq 2$ distinct elements $a_i \in A$ such that

$$a_1 \preceq a_2 \preceq a_3 \preceq a_4 \preceq \ldots \preceq a_{n-1} \preceq a_n \preceq a_1$$

Proof. Suppose that for some $n \geq 2$ such sequence $a_1 \ldots a_n$ exists.

Recall that the partial order is a transitive, antisymmetric, and refelxive relation.

Partially ordered sets

Theorem. A poset $(A, \preceq)$ has no directed cycles other than self-loops, that is, there is no sequence of $n \geq 2$ distinct elements $a_i \in A$ such that

$$a_1 \preceq a_2 \preceq a_3 \preceq a_4 \preceq \ldots \preceq a_{n-1} \preceq a_n \preceq a_1$$

Proof. Suppose that for some $n \geq 2$ such sequence $a_1 \ldots a_n$ exists.

Recall that the partial order is a transitive, antisymmetric, and refelxive relation.

Since it's transitive: $a_1 \preceq a_2$ and $a_2 \preceq a_3$, therefore $a_1 \preceq a_3$.

Similarly, we prove that $a_1 \preceq a_4$, $a_1 \preceq a_5$, $\ldots$, $a_1 \preceq a_n$.

Thus $a_1 \preceq a_n$ and $a_n \preceq a_1$.

But $\preceq$ is antisymmetric, and therefore $a_1 = a_n$. This contradicts the supposition that $a_1, \ldots a_n$ are $n \geq 2$ distinct elements! Thus there is no such directed cycle.

Total order

Def. A *total order* is a partial order in which every pair of elements is comparable.

$(A, \preceq)$ is a total order if for every $x, y \in A$, either $x \preceq y$ or $y \preceq x$.

The $\leq$ relation on natural numbers is a total order. However, the "divides" relation on the same set $\mathbb{N}$ is not.

Question: Given a parially ordered set $(A, \preceq)$, can we make a total order $\preceq_T$ that is "compatible" with the given partial order $\preceq$? (Compatible in the sense that the total order never violates the given partial order)

Topological sort

Def. A *topological sort* of a poset $(A, \preceq)$ is a total order $\preceq_T$ s.t.

$$x \preceq y \quad \text{implies} \quad x \preceq_T y.$$

Theorem. Every finite poset has a topological sort.

Lemma. Every finite poset has a minimal element.