Mathematical Foundations

Definitions and (Some) Proof Techniques (Part 2)

Thursday, September 6, 2007

Reading: Stoughton 1.1

CS235 Languages and Automata

Department of Computer Science
Wellesley College

Pairs and Cross Products

(*a*, *b*) denotes a **pair** = an (ordered) sequence of two elements.

- *a* is the first (or left) element of the pair
- *b* is the second (or right) element of the pair.

For any two sets A and B, $A \times B = \{(a,b) \mid a \in A \text{ and } b \in B\}$.
This is called the **cross product** or **Cartesian product** of A and B.

E.g., Suppose Sign = {-,0,+}

Then Bool x Sign = {(T, -), (T, 0), (T, +), (F, -), (F, 0), (F, +)}

The size of a finite set S is written |S|.

What is |A x B| in terms of |A| and |B|?

Tuples and General Cross Products

We can take the cross product of any number of sets.

An element of $A_1 \times A_2 \times \ldots \times A_k$ is called a **k-tuple**.

For small k, k-tuples have special names:

k	k-tuple	k	k-tuiple
2	pair, duple	6	sextuple
3	triple	7	septuple
4	quadruple	8	octuple
5	quintuple		

A^k stands for $A \times A \times \ldots \times A$ (k times).

A^1 is considered a synonym for A.

A^0 is considered a synonym for Unit = {unit} (a 1-element set)

Characters and Strings

For the time being*, we'll define

Char = {a, b, c, … , A, B, C, .. , 0, 1, 2, … }

String = { s | s ∈ $Char^k$ for some k ∈ Nat}

Strings are usually written using double-quote notation rather than as tuples:

"CS235" stands for (C, S, 2, 3, 5)
"a" stands for (a)
"" stands for ()

The **length** of a string is the size of its tuple.

* In Lec. 4, we'll see that Strings can be parameterized over an alphabet.

Binary Relations

A binary relation on A and B is any subset of A x B. Examples:

containsChar = {(s, c) | s ∈ String, c ∈ Char, c is a char in s}

isSqrtOf = {(i, n) | i ∈ Int, n ∈ Nat, i^2 = n}

closeTo = {(a, b) | a, b ∈ Nat and |a – b| ≤ 2}

sameLen = {(s, t) | s, t ∈ String, length(s) = length(t)}

If R is a binary relation, (a, b) ∈ R is often abbreviated as a R b (*infix notation*) or R(a,b) (*prefix notation*). E.g.

"aqua" containsChar a	*or*	containsChar("aqua", a)
-3 isSqrtOf 9	*or*	isSqrtOf(-3, 9)
5 closeTo 3	*or*	closeTo(5, 3)

A binary relation on A is any subset of A x A.
E.g., closeTo on Nat, sameLen on String

Relations can be generalized to any number of sets.

Some Binary Relation Definitions

The **inverse** of a binary relation R on A and B is the relation R^{-1} = {(b,a) | b ∈ B, a ∈ A, and (a, b) ∈ R).

E.g. a $containsChar^{-1}$ "aqua", 9 $isSqrtOf^{-1}$ -3

A binary relation R on A is:

- **reflexive** iff a R a for all a ∈A.
- **symmetric** iff x R y implies y R x.
- **transitive** iff (x R y and y R z) implies x R z.

An **equivalence relation** is one that's reflexive, symmetric, and transitive.

Relation	reflexive?	symmetric?	transitive?	equiv. rel.?
= on Int				
< on Int				
closeTo				
sameLen				

Closures of Binary Relations

Suppose R is a binary relation on A.

The **reflexive closure** of R is R U {(a, a) | a ∈ A}.

The **transitive closure** of R is the smallest superset of R that's transitive. I.e., it is the smallest relation T such that R ⊆ T and (x, y) ∈ T and (y, z) ∈ T implies (x, z) ∈ T.

The **reflexive transitive closure** of R is the reflexive closure of the transitive closure of R.

E.g. Simple = {(0,5), (5, 3), (3, 8)} (A = {0, 3, 5, 8})
isOneLess = {(a, b) | a, b ∈ Int and a = b - 1}

Relation	Refl. Clos.	Trans. Clos.	Refl. Trans. Clos,
Simple			
isOneLess			

Functions

domain codomain graph

A **function** is a triple (A , B , relation R on A x B) such that (a R b_1 and a R b_2) implies $b_1 = b_2$. I.e., an input can go to at most one output.

A function is **defined at an input** a ∈ A if there is b ∈ B s.t a R b. In this case we write f(a) = b; otherwise f(a) = *undefined*.

A function is **total** iff it is defined at all inputs in its domain. Otherwise it is **partial**. We focus on total functions in this course; assume all functions are total unless stated otherwise.

E.g. Sqr = (Int, Nat, {(i, i^2) | i ∈ Int}) is a total function
NatSqrt = (Nat, Nat, {(n^2, n) | n ∈ Nat}) is a partial function

The **image** of a function f is all the elements in the codomain that are "hit" by applying f to some input:

image(f) = {f(a) | a ∈ domain(f) and f(a) is defined}

What are the images of Sqr and NatSqrt?

Math Functions vs. Programming Functions

A programming language function/procedure/method can always be described by a rule specifying to map input to output. E.g.

public boolean isEven (int n) { return (n % 2) = 0; }

All programming functions can be modeled by mathematical functions. For simple ones like isEven, the correspondence is easy

isEven = (Int, Bool, { (i, (i mod 2) = 0) | i ∈ Int })

but even this isn't quite right, since it glosses over the finiteness of Java integers. Moreover, it takes much care to handle recursion, nontermination, errors, side effects, control transfers, etc. This is the domain of **denotational semantics**.

Even more problematic, we'll see in this course that there are mathematical functions that can't be expressed by any rule. This is the essence of uncomputability.

Even so, when math functions **can** be expressed as a rule its handy to write them that way. E.g., $sqr \in (Int \rightarrow Nat)$; $sqr(x) = x^2$

means $sqr = (Int, Nat, \{ (x, x^2) \mid x \in Int \})$

Function Spaces $A \rightarrow B$

$A \rightarrow B$ denotes the set of all total functions from A to B (i.e., with domain A and codomain B).

Recall Bool = {T,F} and Sign = {-, 0, +}

So Bool → Bool is { (Bool, Bool, {(T,T), (F, T)}),
(Bool, Bool, {(T,T), (F, F)}),
(Bool, Bool, {(T, F), (F, T)}),
(Bool, Bool, {(T, F), (F, F)}) }

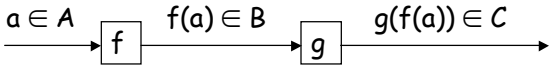
What is Sign → Bool?

What is $|A \rightarrow B|$ in terms of $|A|$ and $|B|$?

Function Composition

If $f \in A \to B$ and $g \in B \to C$ then the composition

$$(g \circ f) \in A \to C \;=\; (A, C, \{ (a, g(f(a))) \mid a \in A \})$$



Function composition is associative: $f \circ (g \circ h) = (f \circ g) \circ h$.

Is it commutative?

For every set A, there is an **identity function** on A:

$$id_A = (A, A, \{ (a, a) \mid a \in A \})$$

Identity functions are the identities of composition:

$$f \circ id_A = f = id_B \circ f$$

Composition Examples

len = (String, Nat, { (s, length(s)) | s ∈ String)})

inc = (Nat, Nat, { (n, n + 1) | n ∈ Nat})

dbl = (Nat, Nat, { (n, n * 2) | n ∈ Nat})

isEvenNat = (Nat, Bool, { (n, (n mod 2) = 0) | n ∈ Nat})

What are the following compositions?

isEvenNat o len =

dbl o inc =

inc o dbl =

isEvenNat o dbl o len =

Function Iteration

Suppose $f \in A \to A$. Then f can be composed with itself.

Define

$f^0 = id_A$

$f^n = f \circ f^{n-1} = f^{n-1} \circ f$

E.g.

$inc^n =$

$dbl^n =$

Injections, Surjections, and Bijections

A function f is **injective** (or **one-to-one**) iff no distinct inputs map to the same output.
I.e., f(x) = f(y) implies x = y.

A function f is **surjective** iff every element in the codomain is hit by some input.
I.e. image(f) = codomain(f).

A function f is **bijective** iff it is injective and surjective.
Equivalently, f in $A \to B$ is bijective if there is an inverse function f^{-1} in $B \to A$ such that $f^{-1} \circ f = id_A$ and $f \circ f^{-1} = id_B$.

The Pigeon-Hole Principle

The **pigeon-hole principle** says that if *p* pigeons are put into less than *p* holes, some hole must end up with more than one pigeon in it.

This principle is often invoked in proofs involving counting arguments.

Suppose A and B are finite sets and |A| > |B|.

- Can a function in A → B be injective? Explain.
- Can a function in A → B be surjective ? Explain.
- Can a function in B → A be injective? Explain.
- Can a function in B → A be surjective? Explain.

Set Sizes

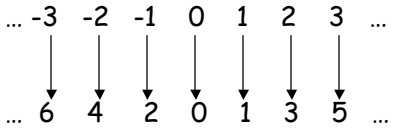
Two sets A and B have the same size (written A ≅ B) if there is some bijection in A → B.

Example 1: Bool ≅ {0,1} by the bijection (Bool, {0,1}, {(T,0), (F,1)})

If A and B are finite sets (|A| = |B|) ⇔ (A ≅ B)

Example 2: A x (B x C) ≅ (A x B) x C by the bijection whose graph has pairs of the form ((a,(b,c)), ((a,b),c)). A, B, and C are not required to be finite! This bijection allows us to treat these two products effectively interchangeably, giving rise to a kind of associativity.

Example 3: Int ≅ Nat by the pictured bijection. How would you define this bijection formally? This is an example of **proof by construction**.



Countable and Uncountable Sets

A set S is

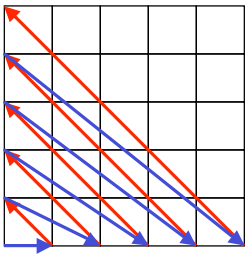
- **finite** iff S ≅ {1, 2, …, n} for some n.
- **infinite** iff S is not finite.
- **countably infinite** iff S ≅ Nat.
- **countable** iff S is finite or countably infinite.
  I.e., there is a procedure for enumerating all the elements of S.
- **uncountable** iff S is not countable

We've seen that Bool, Nat, and Int are countable.

Now we'll see that (1) Rat is countable and
(2) Real is uncountable.

Rat is Countable

Key idea: can enumerate Nat x Nat as follows:



Mopping up:

- Need to eliminate duplicates, e.g., 1/2= 2/4
- Need to handle negative rationals
  (as in showing Int countable).

Real is Uncountable: Diagonalization

Key idea: use a special form of proof by contradiction known as **diagonalization**.

Assume that [0,1) ⊆ Real is countable and derive a contradiction.

If [0,1) is countable, there must be a bijection f ∈ Nat → [0,1) that enumerates all real numbers between 0 (inclusive) and 1 (exclusive). I.e., if r ∈ $[0,1)$, then there is an n ∈ Nat s.t. f(n) = r.

Draw a table of f whose rows are f(n) and whose columns show the digits after the decimal point for each number.

f(0)	1	4	1	5
f(1)	7	3	8	2
f(2)	5	4	9	6
f(3)	8	2	7	3

…

⋮

Diagonalization Continued

f(0)	**1**	4	1	5
f(1)	7	**3**	8	2
f(2)	5	4	**9**	6
f(3)	8	2	7	**3**

…

⋮

Focus on the diagonal entries, and construct a number whose decimal representation differs from every position in the diagonal*. E.g., .2786 …

Any such number is **not** a row in the table and so is not in the image of f. Thus, the assumption that f is a bijection is wrong! ✗

A similar argument can be use to show P(Nat) and Nat → Bool are uncountable.

Diagonalization is the heart of the halting theorem proof.

* For technical reasons, should not use 0 or 9 in the constructed number.

My Functions vs. Stoughton's

Stoughton defines a function as any relation R on A and B in which (x R y and x R z) implies y = z.

If R is a function, then

domain(R) = {a | a ∈ A and a R b for some b ∈ B}

codomain(R) = {b | b ∈ B and a R b for some a ∈ A}

Stoughton uses **range** for codomain. In the literature range is ambiguous: it can mean codomain or image. But in Stoughton's definition, these coincide.

In Stoughton's approach:

- functions are always total;
- functions are always surjective.

This makes it (1) harder to define partial functions and (2) harder to define composition. E.g., what is inc o sqr?