LECTURE 9: FUNCTIONS

Software Engineering

Mike Wooldridge

1 Cartesian Products

- As defined earlier, a set is an *unstructured* object: the order in which elements occur in a set is not important.
- However, many objects in formal system specification require some structure or ordering — otherwise how could we have things like Modula-2 `RECORD`s or C structures?
- *Cartesian products* are one way of making objects which have structure.

- Suppose that

  $$\begin{array}{l} A : \mathbb{P}\, T_1 \\ B : \mathbb{P}\, T_2 \end{array}$$

  (i.e., A is a subset of T_1 and B is a subset of T_2).

  The the cartesian product of A and B is given by the expression

  $$A \times B$$

  and is a set containing all the *ordered pairs* whose first element comes from set A and whose second element comes from set B.

- EXAMPLE. If

  $$\begin{array}{l} A == \{1, 2\} \\ B == \{3, 4\} \end{array}$$

  then

  $$A \times B = \{(1, 3), (1, 4), (2, 3), (2, 4)\}.$$

- An ordered pair is an example of an n-tuple; in this case $n = 2$.
- We list the components of an n-tuple in parentheses.
- Cartesian products are not restricted to just 2 sets — we can have as many as we wish.
- **Definition:** If

  $$S_1, \ldots, S_n$$

  are arbitrary sets, then

  $$S_1 \times \cdots \times S_n$$

  is the set of n-tuples over $S_1, \ldots, S_n$:

  $$\begin{aligned} & S_1 \times \cdots \times S_n == \\ & \quad \{(e_1, \ldots, e_n) \mid e_1 \in S_1 \wedge \cdots \wedge e_n \in S_n\}. \end{aligned}$$

- Things to note about cartesian products:
  - $\#(S_1 \times \cdots \times S_n) = \#S_1 * \cdots * \#S_n$
  - $\neg \forall S_1, S_2 : \mathit{Set} \bullet (S_1 \times S_2) = (S_2 \times S_1)$ (i.e., the cartesian product operation does not commute).
- Finally, let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be an indexed set of sets; then the cartesian product over its component sets is often written:

  $$\Pi_{i \in 1..n}\, S$$

  or just

  $$\Pi\, \mathcal{S}.$$
- Cartesian products are sometimes called cross products.

2 Functions

- Functions are mathematical objects that *take some arguments* and *return some values*.
- One *model* for functions is as a set of *ordered pairs*.
- EXAMPLE. Imagine a function in a Modula-2 program that takes as its sole argument a name representing somebody in a computer department, and returns as its sole result their phone number:

  ```
  PROCEDURE PN(n: Name): PhoneNum
  ```

  So that

  ```
  PN('mike') = 1531
  PN('eric') = 1489
  ```

  We can represent this function as the set

  $$PN == \{(mike, 1531), (eric, 1489)\}$$

- If *PN* is so defined, then we say $PN(mike) = 1531$, and $PN(eric) = 1489$.
- QUESTION: Can we define the set of functions from T_1 to T_2 as the set of ordered pairs $T_1 \times T_2$?
- ANSWER: No — this doesn't work: consider the set

  $$PN == \{(mike, 1531), (mike, 1455)\}$$

  what is $PN(mike)$ defined to be here?
- Functions have a *uniqueness* property: *every possible input to the function must have at most* one *associated output.*

- Note that it is *is* possible for two inputs to map to the *same* output.
- What happens when we try to put a value into a function when there is no corresponding output listed? If

  $$\mathit{PN} == \{(\mathit{mike}, 1531)\}$$

  then $\mathit{PN}(\mathit{eric}) =?$

  In this case we say that the function is *undefined* for that value.

3 Domain and Range

- There are two important sets associated with a function:
  - *domain*: the set representing all input values for which the function is defined;
  - *range*: the set representing all outputs of the function that correspond to a defined input.
- **Definition:** If f is an arbitary function then

  $$\operatorname{dom} f$$

  is an expression returning the domain of f and

  $$\operatorname{ran} f$$

  is an expression returning its range.
- EXERCISE. Using set comprehension, define the domain and range of a function f which maps values from T_1 to T_2.

  SOLUTION.

  $$\operatorname{dom} f == \{x : T_1 \mid \exists y : T_2 \bullet (x, y) \in f\}$$
  $$\operatorname{ran} f == \{x : T_2 \mid \exists y : T_1 \bullet (y, x) \in f\}$$

- EXAMPLE. If

  $$PN == \{(eric, 1489), (mike, 1531)\}$$

  then

  $$\mathrm{dom}\, PN = \{eric, mike\}$$

  and

  $$\mathrm{ran}\, PN = \{1531, 1489\}$$

- Theorems about domain and range:

$$\begin{array}{rcl}
\#\,\mathrm{dom} f & \geq & \#\,\mathrm{ran} f \\
\mathrm{dom}(f \cup g) & = & (\mathrm{dom} f) \cup (\mathrm{dom}\, g) \\
\mathrm{ran}(f \cup g) & = & (\mathrm{ran} f) \cup (\mathrm{ran}\, g) \\
\mathrm{dom}(f \cap g) & \subseteq & (\mathrm{dom} f) \cap (\mathrm{dom}\, g) \\
\mathrm{ran}(f \cap g) & \subseteq & (\mathrm{ran} f) \cap (\mathrm{ran}\, g) \\
\mathrm{dom}\, \emptyset & = & \emptyset \\
\mathrm{ran}\, \emptyset & = & \emptyset
\end{array}$$

4 Total and Partial Functions

- The most general kind of functions we consider are *partial functions*.
- **Definition:** If f is a function from T_1 to T_2, then f is a partial function. The set of all partial functions from T_1 to T_2 is given by the expression

  $$T_1 \nrightarrow T_2.$$

- Note that:
  - $\emptyset \in T_1 \nrightarrow T_2$ (i.e, the emptyset is a partial function).
  - if $f \in T_1 \nrightarrow T_2$ then f may be undefined for some value in T_1.

- Some partial functions have the property of being defined for *all* potential input values: these are *total* functions.
- **Definition:** If $f \in T_1 \nrightarrow T_2$ and $\mathrm{dom} f = T_1$, then f is said to be a *total function* from T_1 to T_2. The set of total functions from T_1 to T_2 is given by the expression:

  $$T_1 \to T_2.$$

- EXERCISE. Define the set $T_1 \to T_2$ using set comprehension.

  SOLUTION.

  $$\begin{array}{l} T_1 \to T_2 == \\ \quad \{f : T_1 \nrightarrow T_2 \mid \mathrm{dom} f = T_1\} \end{array}$$

- QUESTION: What happens if a function takes *more than one* argument?

  ANSWER: Then we say that the function takes just one input, from the cartesian product of the input argument types.

- EXAMPLE. The function *plus* takes two integers as inputs, adds them together and returns the result;

  $$plus : \mathrm{Z} \times \mathrm{Z} \to \mathrm{Z}$$

- The expression

  $$f : D_1 \times \cdots \times D_m \to R_1 \times \cdots \times R_n$$

  which specifies the type of the function f is called the *signature* of f.

5 Properties of Functions

5.1 Injections

- **Definition:** A function is *one-to-one* iff every element in the domain maps to a different element in the range. One-to-one functions are also called *injections*.
- EXAMPLES. The following is an injection:

  $$\{(\mathit{mike}, 1531), (\mathit{eric}, 1489)\}$$

  whereas the following is not:

  $$\{(\mathit{mike}, 1531), (\mathit{eric}, 1531)\}$$

5.2 Surjections

- **Definition:** A function f is *onto* iff every possible element $y \in \operatorname{ran} f$ has some corresponding value $x \in \operatorname{dom} f$ such that $f(x) = y$.
- EXAMPLE. Suppose

  $$\begin{array}{l} T_1 == \{a, b, c, d\} \\ T_2 == \{e, f, g\} \end{array}$$

  $$\begin{array}{l} f_1 : T_1 \pfun T_2 \\ f_2 : T_1 \pfun T_2 \end{array}$$

  Then

  $$f_1 == \{(a, e), (b, f), (c, g)\}$$

  is a surjection; but

  $$f_2 == \{(a, e), (b, f)\}$$

  is not a surjection, as there is no value $x \in \operatorname{dom} f_2$ such that $f_2(x) = g$.
- *Do not confuse surjections with total functions.*

- Finally, if a function is both an injection and a surjection, then it is called a *bijection*.
- There are operators for building combinations of types:

constructor	returns
$\nrightarrow$	partial functions
$\rightarrow$	(total) functions
$\rightarrowtail\!\!\!\!\!\!\!+$	partial injections
$\rightarrowtail$	(total) injections
$\twoheadrightarrow\!\!\!\!\!\!\!+$	partial surjections
$\twoheadrightarrow$	(total) surjections
$\rightarrowtail\!\!\!\!\!\!\!\twoheadrightarrow$	bijections

6 The Maplet Notation

- A more convenient way of writing the function

  $$\{(\mathit{mike}, 1531), (\mathit{eric}, 1489)\}$$

  is to write

  $$\{\mathit{mike} \mapsto 1531, \mathit{eric} \mapsto 1489\}$$

- The symbol $\mapsto$ is called the *maplet arrow*: the expression $\mathit{mike} \mapsto 1531$ is called a *maplet*.
- (The maplet notation is just Z syntactic sugar.)

7 Manipulating Functions

- As functions are just sets, we can use the apparatus of set theory to manipulate them.
- However, there are certain things we do so often that it is useful to define operators for them.

7.1 Domain Restriction

- Suppose, given our function *PN* which maps a person in a department to their phone number, we wanted to extract another function which just contained the details of the logic group.
- Let *LG* be the set containing names of logic group members.
- Then the following expression will do the trick:

  $$LG \lhd PN$$

- $\lhd$ is the *domain restriction* operator.
- **Definition:** Suppose f is a function

  $$f : T_1 \pfun T_2$$

  and S is a set

  $$S : \mathbb{P}\, T_1$$

  then

  $$S \lhd f$$

  is an expression which returns the function obtained from f by removing from it all maplets $x \mapsto y$ such that $x \notin S$.

- EXAMPLE. Let

$$\begin{aligned} PN \;==\; & \{mjw \mapsto 1531, \\ & en \mapsto 1488, \\ & ajt \mapsto 1777\} \end{aligned}$$

and

$$\begin{aligned} & S_1 == \{mike, en\} \\ & S_2 == \{ajt\} \end{aligned}$$

then

$$\begin{aligned} & S_1 \lhd PN = \{mjw \mapsto 1531, en \mapsto 1488\} \\ & S_2 \lhd PN = \{ajt \mapsto 1777\} \end{aligned}$$

- EXERCISE. Define, by set comprehension, the $\lhd$ operator.

$$\begin{aligned} S \lhd f \;==\; & \{x : T_1;\; y : T_2 \mid \\ & (x \in T_1) \wedge (x \mapsto y) \in f \\ & \bullet\, x \mapsto y\} \end{aligned}$$

- Theorems about domain restriction:

$$\begin{aligned} \mathrm{dom}(S \lhd f) &= S \cap \mathrm{dom} f \\ S \lhd f &\subseteq f \\ \emptyset \lhd f &= \emptyset \end{aligned}$$

7.2 Range Restriction

- Just as we can restrict the domain of a function, so we can restrict its range.
- **Definition:** Suppose f is a function

  $$f : T_1 \pfun T_2$$

  and S is a set

  $$S : \mathbb{P}\, T_2$$

  then

  $$f \rhd S$$

  is an expression which returns the function obtained from f by removing from it all maplets $x \mapsto y$ such that $y \notin S$.
- Given PN as previously defined, and

  $$S_1 == \{1531, 1488\}$$
  $$S_2 == \{1777\}$$

  then

  $$f \rhd S_1 = \{mike \mapsto 1531, en \mapsto 1488\}$$
  $$f \rhd S_2 = \{ajt \mapsto 1777\}.$$
- EXERCISE. Define $\rhd\ldots$

7.3 Domain Subtraction

- Suppose we want to take *PN* and *remove* from it all members of the logic group.
- If *LG* is the set containing the logic group, then

  $$LG \ntriangleleft PN$$

  is an expression that will do the trick.
- $\ntriangleleft$ is the *domain subtraction* operator.

  (Also called domain anti-restriction.)

- EXAMPLE. Given *PN* as previously defined, and

  $$S == \{mikew\}$$

  then

  $$S \ntriangleleft PN = \{en \mapsto 1488, ajt \mapsto 1777\}.$$

- **Definition:** Suppose f is a function

  $$f : T_1 \pfun T_2$$

  and S is a set

  $$S : \mathbb{P}\, T_1$$

  then

  $$S \ntriangleleft f$$

  is an expression which returns the function obtained from f by removing from it all maplets $x \mapsto y$ such that $x \in S$.

- EXERCISE. Define $\ntriangleleft$ — you don't need a set comprehension.

  $$S \ntriangleleft f == (\mathrm{dom} f \setminus S) \lhd f$$

7.4 Range Subtraction

- The range subtraction operator is $\nrres$.
- EXERCISE. Given *PN* as previously defined, and

  $$S = \{1531, 1488\}$$

  what does

  $$PN \nrres S$$

  evaluate to?
- EXERCISE. Define $\nrres$ …

  $$f \nrres S == f \rres (\operatorname{ran} f \setminus S).$$

7.5 Function Overriding

- Suppose we have the function *PN* that gives peoples phone numbers, and someone changes their extension number — then we want to reflect this by changing *PN*.

  Given *PN* as previously defined; what expression can we use to change mike's number to 1555?

  $$(PN \setminus \{mike \mapsto 1531\}) \\ \cup\{mike \mapsto 1555\}$$

  Yuk!

- Z provides the $\oplus$ sybol for *function overriding*:

  $$\begin{aligned} PN \oplus \{mike \mapsto 1555\} &= \{mike \mapsto 1555, \\ &\quad en \mapsto 1488, \\ &\quad ajt \mapsto 1777\} \end{aligned}$$

- **Definition:** If

  $$\begin{array}{l} f_1 : T_1 \pfun T_2 \\ f_2 : T_1 \pfun T_2 \end{array}$$

  then

  $$f_1 \oplus f_2$$

  is an expression returning the function that results from overwriting f_1 with f_2:

  $$f_1 \oplus f_2 == (\mathrm{dom}(f_2) \ndres f_1) \cup f_2.$$