More on Event-B: Functions

Modified and Presented by Andy Edmunds

© Michael Butler

University of Southampton

February 6, 2013

Partial Functions

Special kind of relation: each domain element has at most one range element associated with it.

To declare f as a partial function:

$$\boxed{f \ \in \ X \nrightarrow Y}$$

This says that f is a many-to-one relation

Each domain element is mapped to one range element:

$$x \ \in \ dom(f) \qquad \Longrightarrow \qquad card(\ f[\{x\}]\ ) \ = \ 1$$

More usually formalised as a uniqueness constraint

$$x \mapsto y_1 \ \in \ f \ \ \wedge \ \ x \mapsto y_2 \ \in \ f \qquad \Longrightarrow \qquad y_1 = y_2$$

Function Application

We can use function application for partial functions.

If $x \in \mathrm{dom}(f)$, then we write $\boxed{f(x)}$ for the unique range element associated with x in f.

If $x \notin \mathrm{dom}(f)$, then $f(x)$ is undefined.

If $card(\ f[\{x\}]\ ) > 1$, then $f(x)$ is undefined.

Examples

$$dir1 = \{\ mary \mapsto 398620,\ jim \mapsto 493028,\ jane \mapsto 493028\ \}$$

$$dir2 = \{\ mary \mapsto 287573,\ mary \mapsto 398620,\ jane \mapsto 493028\ \}$$

Types of $dir1, dir2$? $dir1(jim)$? $dir1(sarah)$?
$dir2(mary)$?

Examples

$$
\begin{aligned}
dir1 \;=\; \{\; & mary \mapsto 398620, \\
& jim \mapsto 493028, \\
& jane \mapsto 493028 \;\}
\end{aligned}
\qquad
\begin{aligned}
dir2 \;=\; \{\; & mary \mapsto 287573, \\
& mary \mapsto 398620, \\
& jane \mapsto 493028 \;\}
\end{aligned}
$$

Types of $dir1, dir2$? $\quad dir1(jim)$? $\quad dir1(sarah)$?
$dir2(mary)$?

$$
\begin{aligned}
dir1 &\in Person \nrightarrow Phone \\
dir2 &\notin Person \nrightarrow Phone
\end{aligned}
$$

Examples

$$dir1 = \{ \, mary \mapsto 398620, \; jim \mapsto 493028, \; jane \mapsto 493028 \, \}$$

$$dir2 = \{ \, mary \mapsto 287573, \; mary \mapsto 398620, \; jane \mapsto 493028 \, \}$$

Types of $dir1, dir2$? $dir1(jim)$? $dir1(sarah)$? $dir2(mary)$?

$$\begin{aligned}
dir1 &\in Person \pfun Phone \\
dir2 &\notin Person \pfun Phone \\
dir1(jim) &= 493028 \\
dir1(sarah) &\text{ is undefined} \\
dir2(mary) &\text{ is undefined}
\end{aligned}$$

Well-definedness and application definitions

Expression	Well-definedness condition
$f(x)$	$x \in dom(f) \ \wedge \ f \in X \mathrel{+\!\!\!\rightarrow} Y$

The following definition of function application assumes that $f(x)$ is well-defined:

Predicate	Definition
$y = f(x)$	$x \mapsto y \in f$

Function Operators

All the relational operators can be used on functions (restriction, subtraction, image, composition, etc).

Be careful with some operators!

Suppose that f and g are functions.

- Set Union: $f \cup g$ is a function provided

  $$x \in \mathrm{dom}(f) \ \wedge \ x \in \mathrm{dom}(g) \quad \Longrightarrow \quad f(x) = g(x)$$

  Why?
- Inverse: f^{-1} is not always a function. Why not?
- What about $f ; g$?

Function Overriding

Override f by g $\boxed{f \oplus g}$

f and g must be partial functions of the same type

Override: replace existing mappings with new ones

$$\begin{aligned} dir1 \quad = \quad \{ \ & mary \mapsto 398620, \ john \mapsto 829483, \\ & jim \mapsto 493028, \ jane \mapsto 493028 \ \} \end{aligned}$$

$$dir1 \oplus \{ \ mary \mapsto 674321\} \quad = \quad ?$$

$$dir1 \oplus \{ \ mary \mapsto 674321, \ jane \mapsto 829483\} \quad = \quad ?$$

Function Overriding Definition

Definition in terms of function override and set union:

Function Overriding Definition

Definition in terms of function override and set union:

$$
\begin{aligned}
f \mathbin{\lhd\!\!+} \{a \mapsto b\} &= (\{a\} \mathbin{⩤} f) \cup \{a \mapsto b\} \\
f \mathbin{\lhd\!\!+} g &= (\mathrm{dom}(g) \mathbin{⩤} f) \cup g
\end{aligned}
$$

Birthday Book Example

Birthday book relates people to their birthday.

Each person can have at most one birthday.

People can share birthdays.

sets $\quad PERSON \quad DATE$

variables $\quad birthday$
invariants $\quad birthday \in PERSON \pfun DATE$

initialisation $\quad birthday := \{\}$

Adding and checking birthdays

Add an entry to the directory:

Adding and checking birthdays

Add an entry to the directory:

$$
\begin{array}{lcl}
AddEntry & \widehat{=} & \textbf{any } p, d \textbf{ where} \\
 & & \quad p \in Person \\
 & & \quad p \notin dom(birthday) \\
 & & \quad d \in Date \\
 & & \textbf{then} \\
 & & \quad birthday \; := \; birthday \cup \{p \mapsto d\} \\
 & & \textbf{end}
\end{array}
$$

Check a person's birthday:

Adding and checking birthdays

Add an entry to the directory:

$$
\begin{array}{lcl}
AddEntry & \widehat{=} & \textbf{any}\ p, d\ \textbf{where} \\
& & \quad p \in Person \\
& & \quad p \notin dom(birthday) \\
& & \quad d \in Date \\
& & \textbf{then} \\
& & \quad birthday \ := \ birthday \cup \{p \mapsto d\} \\
& & \textbf{end}
\end{array}
$$

Check a person's birthday:

$$
\begin{array}{lcl}
Check & \widehat{=} & \textbf{any}\ p, d!\ \textbf{where} \\
& & \quad p \in dom(birthday) \\
& & \quad d! = birthday(p) \\
& & \textbf{end}
\end{array}
$$

Modifying a birthday

Modify an entry in the directory:

Modifying a birthday

Modify an entry in the directory:

$$
\begin{array}{lcl}
\mathit{ModifyEntry} & \hat{=} & \textbf{any}\ p, d\ \textbf{where} \\
 & & \quad p \in \mathit{dom}(\mathit{birthday}) \\
 & & \quad d \in \mathit{Date} \\
 & & \textbf{then} \\
 & & \quad \mathit{birthday} \ := \ \mathit{birthday} \Leftarrow \{p \mapsto d\} \\
 & & \textbf{end}
\end{array}
$$

Syntactic shorthand:

$$
\begin{array}{lcl}
\mathit{ModifyEntry} & \hat{=} & \textbf{any}\ p, d\ \textbf{where} \\
 & & \quad p \in \mathit{Person} \\
 & & \quad d \in \mathit{Date} \\
 & & \textbf{then} \\
 & & \quad \mathit{birthday}(p) \ := \ d \\
 & & \textbf{end}
\end{array}
$$

Function inverse

Check birthdays on a particular date:

$$
\begin{array}{lll}
\mathit{Who} & \hat{=} & \textbf{any } d, ps! \textbf{ where} \\
 & & \quad d \in \mathit{Date} \\
 & & \quad ps! = \mathit{birthday}^{-1}(d) \\
 & & \textbf{end}
\end{array}
$$

- Is this mathematically valid?

Function inverse

Check birthdays on a particular date:

$$\begin{array}{lll} Who & \widehat{=} & \textbf{any } d, ps! \textbf{ where} \\ & & \quad d \in Date \\ & & \quad ps! = birthday^{-1}(d) \\ & & \textbf{end} \end{array}$$

- Is this mathematically valid?
- No: $birthday^{-1}$ might not be a function.

Function inverse

$birthday^{-1}$ is a relation:

$$birthday^{-1} \quad \in \quad Date \leftrightarrow Person$$

Check birthdays on a particular date:

$$\begin{array}{lll} Who & \hat{=} & \textbf{any } d, ps! \textbf{ where} \\ & & \quad d \in Date \\ & & \quad ps! = birthday^{-1}[\{d\}] \\ & & \textbf{end} \end{array}$$

Alternative:

$$\begin{array}{lll} Who & \hat{=} & \textbf{any } d, ps! \textbf{ where} \\ & & \quad d \in Date \\ & & \quad ps! = dom(birthday \rhd \{d\}) \\ & & \textbf{end} \end{array}$$

Adding the domain as an explicit variable

variables $\quad birthday, person$

invariants

$$
\begin{aligned}
birthday &\in PERSON \pfun DATE \\
person &\subseteq PERSON \\
person &= dom(birthday)
\end{aligned}
$$

initialisation $\quad birthday := \{\} \qquad person := \{\}$

Total Functions

A total function is a special kind of partial function. To declare f as a total function:

$$\boxed{f \;\in\; X \to Y}$$

This means that f is well-defined for every element in X, i.e., $f \;\in\; X \to Y$ is shorthand for

$$f \;\in\; X \prarr Y \quad \wedge \quad dom(f) = X$$

Modelling with Total functions

We can re-write the invariant for the birthday book to use total functions:

variables $birthday, person$
invariants

$$\begin{array}{lcl} person & \subseteq & PERSON \\ birthday & \in & person \rightarrow DATE \end{array}$$

Using the total function arrow means that we don't need to explicitly specify that $dom(birthday) = person$.

We can use $person$ as a guard instead of $dom(birthday)$:

$$\begin{array}{lcl} Check & \widehat{=} & \textbf{any } p, d! \textbf{ where} \\ & & \quad p \in person \\ & & \quad d! = birthday(p) \\ & & \textbf{end} \end{array}$$

AddEntry needs to be modified

Add an entry to the directory:

$$
\begin{array}{lll}
AddEntry & \hat{=} & \textbf{any } p, d \textbf{ where} \\
& & \quad p \in PERSON \\
& & \quad p \notin person \\
& & \quad d \in DATE \\
& & \textbf{then} \\
& & \quad birthday := birthday \cup \{p \mapsto d\} \\
& & \quad person := person \cup \{p\} \\
& & \textbf{end}
\end{array}
$$

Recap

- Function is a special case of a relation.
- Many-to-one: each domain element mapped to a unique range element.
- Relation operators apply – with caution!
- Funtion override.
- Total functions

Secure database example

We consider a secure database. Each object in the database has a odata component.

Each object has a classification between 1 and 10.

Users of the system have a clearance level between 1 and 10.

Users can only read and write objects whose classification is no greater than the user's clearance level.

What are the *types*, *variables*, *events*?

Types and variables

sets $\quad OBJECT \quad USER$
variables $\quad object,\ user,\ class,\ clear$

Types and variables

sets $\quad OBJECT \quad USER$
variables $\quad object, \ user, \ class, \ clear$
invariants

$$
\begin{array}{lcl}
object & \subseteq & OBJECT \\
user & \subseteq & USER \\
class & \in & object \rightarrow (1..10) \\
clear & \in & user \rightarrow (1..10)
\end{array}
$$

initialisation $\quad object := \{\} \quad user := \{\} \quad class := \{\} \quad clear := \{\}$

Types and variables

sets $OBJECT \quad DATA \quad USER$
variables $object, \; user, \; odata, \; class, \; clear$
invariants

$$
\begin{aligned}
&object \;\subseteq\; OBJECT \\
&user \;\subseteq\; USER \\
&odata \;\in\; object \to DATA \\
&class \;\in\; object \to (1..10) \\
&clear \;\in\; user \to (1..10)
\end{aligned}
$$

The invariant $odata \;\in\; object \to DATA$ means that $odata(o)$ is well-defined whenever $o \in object$. Why is this important?

initialisation

$object := \{\} \quad user := \{\} \quad odata := \{\} \quad class := \{\} \quad clear := \{\}$

Adding users

$$AddUser \quad \widehat{=}$$

Adding users

$$
\begin{array}{l}
\mathit{AddUser} \;\widehat{=} \\
\quad \textbf{any } u, c \textbf{ where} \\
\qquad u \in \mathit{USER} \\
\qquad u \notin \mathit{user} \\
\qquad c \in 1..10 \\
\quad \textbf{then} \\
\qquad \mathit{user} := \mathit{user} \cup \{u\} \\
\qquad \mathit{clear}(u) := c \\
\quad \textbf{end}
\end{array}
$$

The new user must not already exist.
We need to provide the initial clearance level for the new user.

Adding objects

$$AddObject \triangleq$$

Adding objects

$$
\begin{array}{l}
AddObject \;\; \widehat{=} \\
\quad \textbf{any} \;\; o, d, c \;\; \textbf{where} \\
\qquad o \;\; \in \;\; OBJECT \\
\qquad o \;\; \notin \;\; object \\
\qquad d \;\; \in \;\; DATA \\
\qquad c \;\; \in \;\; 1..10 \\
\quad \textbf{then} \\
\qquad object \;\; := \;\; object \cup \{o\} \\
\qquad odata(o) \;\; := \;\; d \\
\qquad class(o) \;\; := \;\; c \\
\quad \textbf{end}
\end{array}
$$

The new object must not already exist.
We need to provide the initial classification level and odata value for the new object.

Reading objects

$$Read \;\; \hat{=}$$

Reading objects

$Read \;\; \hat{=}$

any $u, o, d!$ **where**

$u \in user$	The user must exist
$o \in object$	The object must exist
$clear(u) \geq class(o)$	The clearance must be ok
$d! = odata(o)$	The odata associated with the object

end

Writing objects

$$Write \quad \hat{=}$$

Writing objects

$$
\begin{array}{l}
\mathit{Write} \;\; \widehat{=} \\
\qquad \textbf{any}\ u, o, d\ \textbf{where} \\
\qquad\qquad u \in \mathit{user} \\
\qquad\qquad o \in \mathit{object} \\
\qquad\qquad \mathit{clear}(u) \geq \mathit{class}(o) \\
\qquad \textbf{then} \\
\qquad\qquad \mathit{odata}(o) := d \\
\qquad \textbf{end}
\end{array}
$$

The write operation overwrites the odata value associate with the object with a new value.

Changing classification and clearance levels

$$
\begin{array}{l}
\mathit{ChangeClass} \;\; \widehat{=} \\
\quad \textbf{any}\;\; o, c \;\; \textbf{where} \\
\quad\quad o \;\in\; \mathit{object} \\
\quad\quad c \;\in\; 1..10 \\
\quad \textbf{then} \\
\quad\quad \mathit{class}(o) \;:=\; c \\
\quad \textbf{end}
\end{array}
$$

$$
\begin{array}{l}
\mathit{ChangeClear} \;\; \widehat{=} \\
\quad \textbf{any}\;\; u, c \;\; \textbf{where} \\
\quad\quad u \;\in\; \mathit{user} \\
\quad\quad c \;\in\; 1..10 \\
\quad \textbf{then} \\
\quad\quad \mathit{clear}(u) \;:=\; c \\
\quad \textbf{end}
\end{array}
$$

Removing users and objects

$$RemoveUser \quad \widehat{=}$$

Removing users and objects

$$
\begin{array}{l}
\mathit{RemoveUser} \;\; \widehat{=} \\
\quad \textbf{any } u \textbf{ where} \\
\qquad u \in \mathit{user} \\
\quad \textbf{then} \\
\qquad \mathit{user} := \mathit{user} \setminus \{u\} \\
\qquad \mathit{clear} := \{u\} \lhd\!\!\!- \mathit{clear} \\
\quad \textbf{end}
\end{array}
$$

Removing users and objects

$$
\begin{array}{l}
RemoveUser \quad \hat{=} \\
\quad \textbf{any} \;\; u \;\; \textbf{where} \\
\quad\quad u \;\; \in \;\; user \\
\quad \textbf{then} \\
\quad\quad user \;\; := \;\; user \setminus \{u\} \\
\quad\quad clear := \{u\} \lhd clear \\
\quad \textbf{end}
\end{array}
$$

$$RemoveObject \quad \hat{=}$$

Removing users and objects

$RemoveUser \;\; \widehat{=}$
any u **where**
$u \;\in\; user$
then
$user \;:=\; user \setminus \{u\}$
$clear := \{u\} \lhd clear$
end

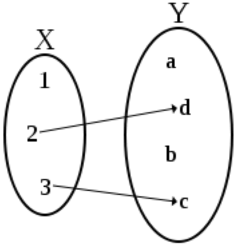
$RemoveObject \;\; \widehat{=}$
any o **where**
$o \;\in\; object$
then
$object \;:=\; object \setminus \{o\}$
$class := \{o\} \lhd class$
$odata := \{o\} \lhd odata$
end

More on Functions

- $\rightarrowtail\!\!\!\!\!+\!\!\!\!\rightarrow$ Partial Injection
- $\rightarrowtail$ Total Injection
- $\mathrel{+\!\!\!\!\twoheadrightarrow}$ Partial surjection
- $\twoheadrightarrow$ Total surjection
- $\rightarrowtail\!\!\!\!\twoheadrightarrow$ Bijection

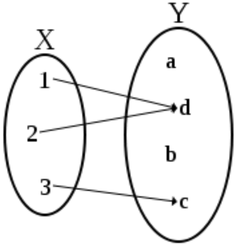
More on Functions

$\nrightarrow$ Partial function



Some domain elements not related to range.

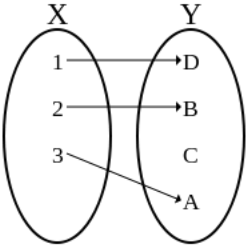
$\rightarrow$ Total function



All domain elements related to range.

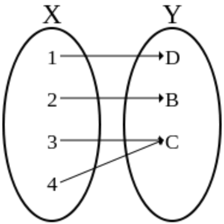
More on Functions

$\rightarrowtail$ Injection



Range elements have a maximum of one element from the domain.

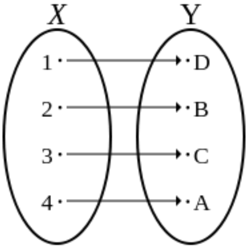
$\twoheadrightarrow$ Surjection



All range elements are related to the domain by at least one domain element.

More on Functions

$\rightarrowtail\!\!\!\!\twoheadrightarrow$ (Total) Bijection



Every element in the domain is paired in a one-one correspondence with a surjective range.

The Lambda Function

From the Rodin handbook:

- $(\lambda p \cdot P | E)$ is a function that maps an input p to a result E such that P holds.
- p is a pattern of identifiers, parentheses and maplets which follow the following rules:
  - An identfier x is a pattern.
  - An identifier x, followed by an *oftype* operator is a pattern
  - A pair $a \mapsto b$ is a pattern if a and b are patterns.
  - (a) is pattern if a is pattern.
  - In the simplest case, p is just an identifier.
- $(\lambda p \cdot P | E) = \{z \cdot P | p \mapsto E\}$
  where z is a list of variables that appear in the pattern p

Lambda Examples

Examples:

- A function *double* that returns the double value of a natural number:

$$double = (\lambda x \cdot x \in \mathbb{N} | 2 * x)$$

- The dot product of two 2-dimensional vectors can be defined by:

$$dotp =$$
$$(\lambda (a \mapsto b) \mapsto (c \mapsto d) \cdot$$
$$a \in \mathbb{Z} \land b \in \mathbb{Z} \land c \in \mathbb{Z} \land d \in \mathbb{Z} | a * c + b * d)$$

Now For Some Practice

Extend the database specification so that each object has an owner.

The clearance associated with that owner must be at least as high as the classification of the object.

Only the owner of an object is allowed to delete it.

A user's clearance level can only be modified to a new level by another user whose clearance level is at least as high as the new clearance level.

What additional variables are required?

What events are affected?