More on Event-B: Functions

© Michael Butler

University of Southampton

May 22, 2010

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:

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

More usually formalised as a uniqueness constraint

$$x \mapsto y_1 \in f \ \wedge \ x \mapsto y_2 \in f \implies y_1 = y_2$$

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

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

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

$$dir1 \in Person \pfun Phone$$

$$dir1(jim) = 493028$$

$$dir1(sarah) \text{ is undefined}$$

$$dir2 \notin Person \pfun Phone$$

$$dir2(mary) \text{ is undefined}$$

Well-definedness and application definitions

Expression	Well-definedness condition
$f(x)$	$x \in dom(f) \ \wedge \ f \in X \nrightarrow Y$

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 $f ; g$?

Birthday Book Example

Birthday book relates people to their birthday.

Each person can have at most one birthday.

People can share birthdays.

sets $PERSON \quad DATE$

variables $birthday$
invariants $birthday \in PERSON \rightarrowtail\!\!\!\!+ DATE$

initialisation $birthday := \{\}$

Adding and checking birthdays

Add an entry to the directory:

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

Check a person's birthday:

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

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:

$$f \oplus \{a \mapsto b\} = (\{a\} \ntriangleleft f) \cup \{a \mapsto b\}$$

$$f \oplus g = (\mathrm{dom}(g) \ntriangleleft f) \cup g$$

Modifying a birthday

Modify an entry in the directory:

$$
\begin{array}{lll}
ModifyEntry & \hat{=} & \textbf{any } p, d \textbf{ where} \\
 & & \quad p \in dom(birthday) \\
 & & \quad d \in Date \\
 & & \textbf{then} \\
 & & \quad birthday := birthday \lhd\!\!\!- \{p \mapsto d\} \\
 & & \textbf{end}
\end{array}
$$

Syntactic shorthand:

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

Function inverse

Check birthdays on a particular date:

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

- Is this mathematically valid?

Function inverse

Check birthdays on a particular date:

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

- Is this mathematically valid?
- No: $\mathit{birthday}^{-1}$ 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:

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

Alternative:

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

Adding the domain as an explicit variable

variables $\quad birthday, person$

invariants

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

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

Total Functions

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

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

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

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

Modelling with Total functions

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

variables $birthday, person$
invariants

$$person \subseteq PERSON$$
$$birthday \in person \rightarrow DATE$$

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 $dom(birthday)$:

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

AddEntry needs to be modified

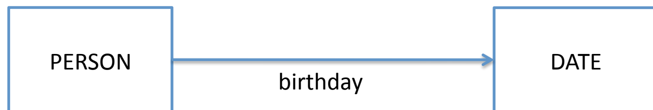
Add an entry to the directory:

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

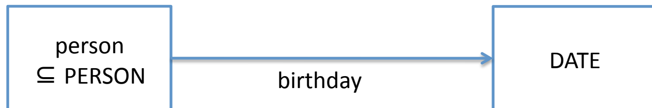
Recap

- 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

Class diagram for the birthday book



Making variable set explicit



Secure database example

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

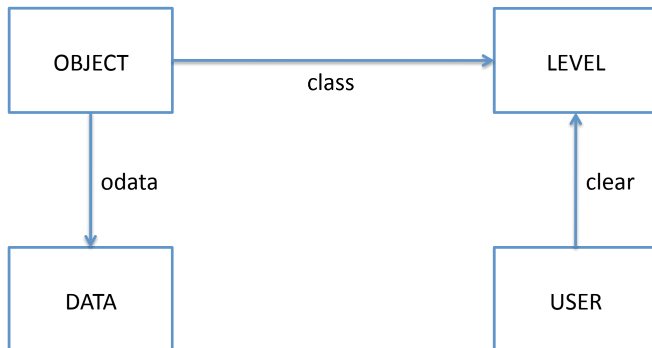
Each object has a classification between 1 and 10.

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

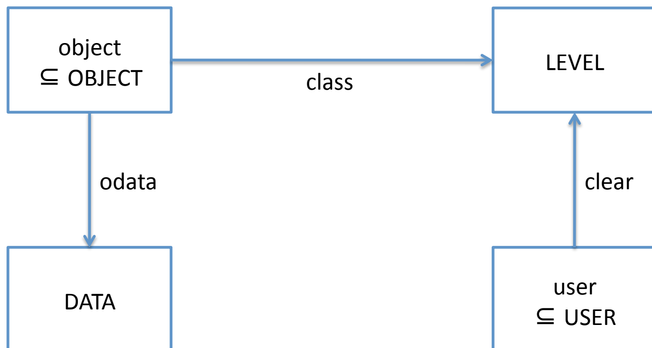
Users can only read and write objects whose classification is no greater than the user's clearance level.

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

Class diagram for secure database



Making variable set explicit



Types and variables

sets $OBJECT \quad DATA \quad USER$
constants $LEVEL$
axioms $LEVEL = 1..10$

variables $object,\ user,\ odata,\ class,\ clear$
invariants

$$
\begin{aligned}
object &\subseteq OBJECT \\
user &\subseteq USER \\
odata &\in object \to DATA \\
class &\in object \to LEVEL \\
clear &\in user \to LEVEL
\end{aligned}
$$

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

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

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

Adding objects

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

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

Reading objects

$$
\begin{array}{lll}
Read & \widehat{=} & \\
& \textbf{any } u, o, result \textbf{ where} & \\
& \quad u \in user & \text{The user must exist} \\
& \quad o \in object & \text{The object must exist} \\
& \quad clear(u) \geq class(o) & \text{The clearance must be ok} \\
& \quad result = odata(o) & \text{The odata associated with the object} \\
& \textbf{end} &
\end{array}
$$

Writing objects

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

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

Changing classification and clearance levels

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

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

Making classification changes more secure

Include constraints on the user who is changing the object classification:

$$
\begin{array}{l}
\mathit{ChangeClass} \ \hat{=} \\
\quad \textbf{any } o, c, u \textbf{ where} \\
\quad\quad o \in \mathit{object} \\
\quad\quad c \in \mathit{LEVEL} \\
\quad\quad \mathit{clear}(u) \geq \mathit{class}(o) \\
\quad\quad \mathit{clear}(u) \geq c \\
\quad \textbf{then} \\
\quad\quad \mathit{class}(o) := c \\
\quad \textbf{end}
\end{array}
$$

Making clearance changes more secure

Include constraints on the user who is changing the object classification:

$$
\begin{array}{l}
\mathit{ChangeClear} \quad \widehat{=} \\
\quad \textbf{any} \; u, a, c \; \textbf{where} \\
\qquad u \in \mathit{user} \\
\qquad a \in \mathit{user} \\
\qquad \mathit{clear}(a) \geq \mathit{clear}(u) \\
\qquad \mathit{clear}(a) \geq c \\
\qquad c \in \mathit{LEVEL} \\
\quad \textbf{then} \\
\qquad \mathit{clear}(u) := c \\
\quad \textbf{end}
\end{array}
$$

Removing users and objects

$$
\begin{aligned}
&RemoveUser \;\; \widehat{=} \\
&\quad \textbf{any } u \textbf{ where} \\
&\quad\quad u \in user \\
&\quad \textbf{then} \\
&\quad\quad user := user \setminus \{u\} \\
&\quad\quad clear := \{u\} \lhd clear \\
&\quad \textbf{end}
\end{aligned}
$$

$$
\begin{aligned}
&RemoveObject \;\; \widehat{=} \\
&\quad \textbf{any } o \textbf{ where} \\
&\quad\quad o \in object \\
&\quad \textbf{then} \\
&\quad\quad object := object \setminus \{o\} \\
&\quad\quad class := \{o\} \lhd class \\
&\quad\quad odata := \{o\} \lhd odata \\
&\quad \textbf{end}
\end{aligned}
$$

Adding object ownership

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?

Class diagram with ownership