Relations and Functions

Michael Butler

SETSS 2016, Chongqing
Requirements for a Buildings Access System

• Specify a system that controls access to a collection of buildings.
• Registered users will have access permission to enter certain buildings.
• A user can only enter buildings that they have access permission for.
• The system should keep track of the location of users.
• The system should manage registration and access permission for users.
Users and Buildings

Carrier sets: USER  BUILDING
Permission

Many-to-many relation
Many-to-one relation
Location conforms to Permission

Location \subseteq Permission
Class diagram abstraction

Many-to-many association

user

permission

location

building

Many-to-one association
Ordered Pairs and Cartesian Products

An ordered pair is an element consisting of two parts: a first part and a second part.

An ordered pair with first part $x$ and second part $y$ is written: $x \mapsto y$

The Cartesian product of two sets is the set of pairs whose first part is in $S$ and second part is in $T$.

The Cartesian product of $S$ with $T$ is written: $S \times T$
Cartesian Products: Definition and Examples

Defining Cartesian product:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \mapsto y \in S \times T$</td>
<td>$x \in S \land y \in T$</td>
</tr>
</tbody>
</table>

Examples:

\[
\{a, b, c\} \times \{1, 2\} = \{ a \mapsto 1, a \mapsto 2, b \mapsto 1, b \mapsto 2, c \mapsto 1, c \mapsto 2 \}
\]

\[
\{a, b, c\} \times \{\} = ?
\]

\[
\{\{a\}, \{a, b\}\} \times \{1, 2\} = ?
\]
Cartesian Products: Definition and Examples

Defining Cartesian product:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a, b, c} \times {1, 2}$</td>
<td>${ a \mapsto 1, a \mapsto 2, b \mapsto 1,$ $b \mapsto 2, c \mapsto 1, c \mapsto 2$ }</td>
</tr>
<tr>
<td>${{a}, {a, b}} \times {1, 2}$</td>
<td>${ {a} \mapsto 1, {a} \mapsto 2,$ ${a, b} \mapsto 1, {a, b} \mapsto 2 }$</td>
</tr>
</tbody>
</table>
Cartesian Product is a Type Constructor

\[ S \times T \] is a new type constructed from types \( S \) and \( T \).

Cartesian product is the type constructor for ordered pairs.

Given \( x \in S, \ y \in T \), we have

\[ x \mapsto y \in S \times T \]

\[ 4 \mapsto 7 \in \ ? \]

\[ \{5, 6, 3\} \mapsto 4 \in \ ? \]

\[ \{4 \mapsto 8, \ 3 \mapsto 0, \ 2 \mapsto 9\} \in \ ? \]
Cartesian Product is a Type Constructor

\[ S \times T \] is a new type constructed from types \( S \) and \( T \).

Cartesian product is the type constructor for ordered pairs.

Given \( x \in S \), \( y \in T \), we have

\[ x \mapsto y \in S \times T \]

\[ 4 \mapsto 7 \in \mathbb{Z} \times \mathbb{Z} \]

\[ \{ 5, 6, 3 \} \mapsto 4 \in \mathcal{P}(\mathbb{Z}) \times \mathbb{Z} \]

\[ \{ 4 \mapsto 8, 3 \mapsto 0, 2 \mapsto 9 \} \in \mathcal{P}(\mathbb{Z} \times \mathbb{Z}) \]
Sets of Order Pairs

A database can be modelled as a set of ordered pairs:

\[
directory = \{ \text{mary} \mapsto 287573, \\
             \text{mary} \mapsto 398620, \\
             \text{john} \mapsto 829483, \\
             \text{jim} \mapsto 398620 \}\n\]

\textit{directory} has type

\[
directory \in \mathbb{P}(Person \times \text{PhoneNum})
\]
A **relation** is a set of ordered pairs.

A relation is a common modelling structure so Event-B has a special notation for it:

\[
T \leftrightarrow S = \mathcal{P}(T \times S)
\]

So we can write:

\[
directory \in Person \leftrightarrow PhoneNum
\]

Do not confuse the arrow symbols:

\[\leftrightarrow\] combines **two sets** to form a **set**.

\[\mapsto\] combines **two elements** to form an **ordered pair**.
Domain and Range

\[
directory = \{ \text{mary} \mapsto 287573, \\
               \text{mary} \mapsto 398620, \\
               \text{john} \mapsto 829483, \\
               \text{jim} \mapsto 398620 \}
\]

\[
dom(directory) = \{ \text{mary}, \text{john}, \text{jim} \}
\]

\[
ran(directory) = \{ 287573, 398620, 829483 \}
\]
Domain and Range Definition

- The **domain** of a relation $R$ is the set of first parts of all the pairs in $R$, written $\text{dom}(R)$.
- The **range** of a relation $R$ is the set of second parts of all the pairs in $R$, written $\text{ran}(R)$.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \in \text{dom}(R)$</td>
<td>$\exists y \cdot x \mapsto y \in R$</td>
</tr>
<tr>
<td>$y \in \text{ran}(R)$</td>
<td>$\exists x \cdot x \mapsto y \in R$</td>
</tr>
</tbody>
</table>
Telephone Directory Model

- Phone directory relates people to their phone numbers.
- Each person can have zero or more numbers.
- People can share numbers.

context PhoneContext
sets Person PhoneNum
end

machine PhoneBook
variables dir
invariants dir \in Person \leftrightarrow PhoneNum

initialisation dir := \{\}
Add an entry to the directory:

\[
AddEntry \triangleq \text{any } p, n \text{ where } \\
p \in \text{Person} \\
n \in \text{PhoneNum} \\
\text{then} \\
dir := dir \cup \{p \mapsto n\} \\
\text{end}
\]
Relational Image

\[
directory = \{ \text{mary} \mapsto 287573, \\
\quad \text{mary} \mapsto 398620, \\
\quad \text{john} \mapsto 829483, \\
\quad \text{jim} \mapsto 398620 \}
\]

Relational image examples:

\[
directory[ \{ \text{mary} \} ] = \{ 287573, 398620 \}
\]
\[
directory[ \{ \text{john}, \text{jim} \} ] = \{ 829483, 398620 \}
\]
Relational Image Definition

Assume \( R \in S \leftrightarrow T \) and \( A \subseteq S \)

The **relational image** of set \( A \) under relation \( R \) is written \( R[A] \)

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y \in R[A] )</td>
<td>( \exists x \cdot x \in A \ \land \ x \mapsto y \in R )</td>
</tr>
</tbody>
</table>
Modelling Queries using Relational Image

Determine all the numbers associated with a person in the directory:

\[
\text{GetNumbers} \triangleq \text{any } p, \text{result where} \\
p \in \text{Person} \\
\text{result} = \text{dir}[ \{p\} ] \\
\text{end}
\]

Determine all the numbers associated with a set of people:

\[
\text{GetMultiNumbers} \triangleq \text{any } ps, \text{result where} \\
ps \subseteq \text{Person} \\
\text{result} = \text{dir}[ \text{ps} ] \\
\text{end}
\]
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:

\[
f \in X \rightarrow Y
\]

This says that \( f \) is a many-to-one relation

Each domain element is mapped to one range element:

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

More usually formalised as a uniqueness constraint

\[
x \mapsto y_1 \in f \quad \land \quad x \mapsto y_2 \in f \quad \implies \quad y_1 = y_2
\]
Function Application

We can use function application for partial functions.

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

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

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

\[ dir_1 = \{ \text{mary} \mapsto 398620, \text{jim} \mapsto 493028, \text{jane} \mapsto 493028 \} \]
\[ dir_2 = \{ \text{mary} \mapsto 287573, \text{mary} \mapsto 398620, \text{jane} \mapsto 493028 \} \]

\[ dir_1 \in \text{Person} \mapsto \text{Phone} \]
\[ dir_1(\text{jim}) = 493028 \]
\[ dir_1(\text{sarah}) \text{ is undefined} \]

\[ dir_2 \notin \text{Person} \mapsto \text{Phone} \]
\[ dir_2(\text{mary}) \text{ is undefined} \]
Well-definedness and application definitions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Well-definedness condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x)$</td>
<td>$x \in \text{dom}(f) \land f \in X \rightarrow Y$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = f(x)$</td>
<td>$x \mapsto y \in f$</td>
</tr>
</tbody>
</table>
Birthday Book Example

Birthday book relates people to their birthday.

Each person can have at most one birthday.

People can share birthdays.

**sets**  \( \text{PERSON} \quad \text{DATE} \)

**variables**  \( \text{birthday} \)

**invariants**  \( \text{birthday} \in \text{PERSON} \rightarrow \text{DATE} \)

**initialisation**  \( \text{birthday} := \{\} \)
Adding and checking birthdays

Add an entry to the directory:

\[
\text{AddEntry} \triangleq \text{any } p, d \text{ where } \\
p \in \text{Person} \\
p \not\in \text{dom}(\text{birthday}) \\
d \in \text{Date} \\
\text{then} \\
\text{birthday} \ := \ \text{birthday} \cup \{p \mapsto d\} \\
\text{end}
\]

Check a person’s birthday:

\[
\text{Check} \triangleq \text{any } p, \text{result} \text{ where } \\
p \in \text{dom}(\text{birthday}) \\
\text{result} = \text{birthday}(p) \\
\text{end}
\]
Domain Restriction

Given \( R \in S \leftrightarrow T \) and \( A \subseteq S \),
the domain restriction of \( R \) by \( A \) is written \( A \triangleleft R \).

Restrict relation \( R \) so that it only contains pairs whose first part is in the set \( A \).

Example:

\[
directory = \{ \text{mary} \mapsto 287573, \text{mary} \mapsto 398620, \\
\text{john} \mapsto 829483, \text{jim} \mapsto 398620 \}
\]

\[
\{\text{john, jim, jane}\} \triangleleft directory = \{ \text{john} \mapsto 829483, \\
\text{jim} \mapsto 398620 \}
\]
Domain Subtraction

Given $R \in S \leftrightarrow T$ and $A \subseteq S$, the domain subtraction of $R$ by $A$ is written $A \triangleleft R$.

Remove those pairs from $R$ whose first part is in $A$.

Example:

\[
directory = \{ \text{mary} \mapsto 287573, \text{mary} \mapsto 398620, \\
\text{john} \mapsto 829483, \text{jim} \mapsto 398620 \}
\]

\[
\{\text{john, jim, jane}\} \triangleleft directory = \{ \text{mary} \mapsto 287573, \\
\text{mary} \mapsto 398620 \}
\]
## Domain and Range, Restriction and Subtraction

Assume $R \in S \leftrightarrow T$ and $A \subseteq S$ and $B \subseteq T$

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Definition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \mapsto y \in A \triangleleft R$</td>
<td>$x \mapsto y \in R \land x \in A$</td>
<td>domain restriction</td>
</tr>
<tr>
<td>$x \mapsto y \in A \triangleleft R$</td>
<td>$x \mapsto y \in R \land x \notin A$</td>
<td>domain subtraction</td>
</tr>
<tr>
<td>$x \mapsto y \in R \triangleright B$</td>
<td>$x \mapsto y \in R \land y \in B$</td>
<td>range restriction</td>
</tr>
<tr>
<td>$x \mapsto y \in R \triangleright B$</td>
<td>$x \mapsto y \in R \land y \notin B$</td>
<td>range subtraction</td>
</tr>
</tbody>
</table>
Removing Entries from the Directory

Remove all the entries associated with a person in the directory:

\[
\text{RemovePerson} \triangleq \quad \text{any } p \text{ where } p \in \text{Person} \\
\text{then} \\
\quad \text{dir} := \{p\} \triangleleft \text{dir} \\
\text{end}
\]

Remove all the entries associated with a number in the directory:

\[
\text{RemoveNumber} \triangleq \quad \text{any } n \text{ where } n \in \text{PhoneNum} \\
\text{then} \\
\quad \text{dir} := \text{dir} \triangleright \{n\} \\
\text{end}
\]
Function Overriding

Override $f$ by $g$ \[ f \leftarrow g \]

$f$ and $g$ must be partial functions of the same type

Override: replace existing mappings with new ones

\[
\text{dir1} = \{ \text{mary} \mapsto 398620, \text{john} \mapsto 829483, \\
\text{jim} \mapsto 493028, \text{jane} \mapsto 493028 \}
\]

\[
\text{dir1} \leftarrow \{ \text{mary} \mapsto 674321, \text{jane} \mapsto 829483 \}
\]

\[
= \{ \text{mary} \mapsto 674321, \text{john} \mapsto 829483, \\
\text{jim} \mapsto 493028, \text{jane} \mapsto 829483 \}
\]
Definition in terms of function override and set union:

\[ f \triangleleft \{a \mapsto b\} = (\{a\} \triangleleft f) \cup \{a \mapsto b\} \]

\[ f \triangleleft g = (\text{dom}(g) \triangleleft f) \cup g \]
Modifying a birthday

Modify an entry in the directory:

\[
\text{ModifyEntry} \triangleq \text{any } p, d \text{ where } \\
p \in \text{dom}(\text{birthday}) \\
d \in \text{Date} \\
\text{then} \\
\text{birthday} \; := \; \text{birthday} \leftarrow \{p \mapsto d\} \\
\text{end}
\]

Syntactic shorthand:

\[
\text{ModifyEntry} \triangleq \text{any } p, d \text{ where } \\
p \in \text{Person} \\
d \in \text{Date} \\
\text{then} \\
\text{birthday}(p) \; := \; d \\
\text{end}
\]
Adding the domain as an explicit variable

variables  \( \text{birthday, person} \)

invariants

\[
\begin{align*}
\text{birthday} & \in \text{PERSON} \rightarrow \text{DATE} \\
\text{person} & \subseteq \text{PERSON} \\
\text{person} & = \text{dom} (\text{birthday})
\end{align*}
\]

initialisation  \( \text{birthday} := \{\} \quad \text{person} := \{\} \)
A total function is a special kind of partial function. To declare \( f \) as a total function:

\[
f \in X \rightarrow Y
\]

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

\[
f \in X \rightarrow Y \land dom(f) = X
\]
Modelling with Total functions

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

variables \( birthday, person \)

invariants

\[
\begin{align*}
\text{person} & \subseteq \text{PERSON} \\
birthday & \in \text{person} \rightarrow \text{DATE}
\end{align*}
\]

Using the total function arrow means that we don’t need to explicitly specify that \( \text{dom}(birthday) = \text{person} \).

We can use \( \text{person} \) as a guard instead of \( \text{dom}(birthday) \):

\[
\text{Check} \triangleq \forall p, \text{result} \ \text{where} \\
p \in \text{person} \\
\text{result} = \text{birthday}(p) \\
\text{end}
\]
AddEntry needs to be modified

Add an entry to the directory:

\[
\text{AddEntry} \triangleq \text{any } p, d \text{ where } \\
p \in \text{PERSON} \\
p \notin \text{person} \\
d \in \text{DATE} \\
\text{then} \\
\text{birthday} := \text{birthday} \cup \{p \mapsto d\} \\
\text{person} := \text{person} \cup \{p\} \\
\text{end}
\]
Requirements for a Buildings Access System

- Specify a system that controls access to a collection of buildings.
- Registered users will have access permission to enter certain buildings.
- A user can only enter buildings that they have access permission for.
- The system should keep track of the location of users.
- The system should manage registration and access permission for users.

Types? Variables? Invariants? Events?
Buildings Access System

- Types: \textit{USER}, \textit{BUILDING}
- Variables: \textit{register}, \textit{permission}, \textit{location}
- Invariants:
  - \textit{register} \subseteq \textit{USER} \quad // register is a set of users
  - \textit{permission} \in \textit{USER} \leftrightarrow \textit{BUILDING}
    \quad // relates users to the buildings they can access
  - \text{dom}(\textit{permission}) \subseteq \textit{register}
    \quad // only register users may have permissions
  - \textit{location} \in \textit{USER} \leftrightarrow \textit{BUILDING}
    \quad // user is located in at most one building
  - \textit{location} \subseteq \textit{permission}
    \quad // user located in a building must have permission
    \quad // for that building
Buildings Access System

Events:
- RegisterUser, DeRegisterUser
- AddPermission, RevokePermission
- EnterBuilding, LeaveBuilding