Lecture 4: Outline

- Reminder: relations
- More operations on relations
- Example: a printer access system
- Total and partial functions (as a special kind of relations)
- Example: the vending machine revisited

Relations (reminder)

- A relation R between sets S and T can be represented as a set of pairs (s, t) representing those elements of S and T that are related
- Mathematically, a relation between sets S and T is a subset of $S \times T$ (or, equivalently, a member of $\mathbb{P}(S \times T)$)
- Note: $S \times T$ – all possible pair combinations between elements of S and T
- Note: $\mathbb{P}(S)$ – all possible subsets of S. Hence, $x \in \mathbb{P}(S)$ means that x is some subset of S, i.e., $x \subseteq S$
- $S \leftrightarrow T$ is a shorthand for $\mathbb{P}(S \times T)$

Relations (reminder)

- 3 equivalent statements saying that R is a relation between S and T:
  $R \in S \leftrightarrow T, \qquad R \in \mathbb{P}(S \times T), \qquad R \subseteq S \times T$
- The *domain* of a relation $R \in S \leftrightarrow T$ is the subset of elements of S that are related to something in T
- Dually, the domain *range* of a relation $R \in S \leftrightarrow T$ is the subset of elements of T that are related to something in S
- We can filter relations, focusing on the relation domain (domain restriction $S \lhd R$ or domain subtraction $S \mathbin{⩤} R$) or the relation range (range restriction $R \rhd S$ or range subtraction $R \mathbin{⩥} S$)

Basic operations with relations

- Initialising (for $R \in S \leftrightarrow T$):
  $R := \{s_1 \mapsto t_1,\ s_2 \mapsto t_2,\ ...\}$ or $R := S \times \{t_0\}$
  where $s_i \mapsto t_i$ are constructed pairs from $S \leftrightarrow T$
- Adding, merging elements (using set operations):
  $R := R \cup \{s \mapsto t\}$ or $R := R \cup Q$
  where Q is a relation of the same type, i.e., $Q \in S \leftrightarrow T$
- Checking membership or inclusion:
  $s \mapsto t \in R$ or $R \subseteq Q$
- Removing elements (filtering on the relation domain):
  $R := \{s_1, s_2, ...\} \lhd R$ or $R := \{s_1, s_2, ...\} \ntriangleleft R$
- Removing elements (filtering on the relation range):
  $R := R \rhd \{t_1, t_2, ...\}$ or $R := R \ntriangleright \{t_1, t_2, ...\}$

More operations on relations

- Inverse relation R^{-1} (ascii R~).
  Includes all such $(x \mapsto y)$ that $(y \mapsto x) \in R$
- Example: for the relation
  $owns\_camera \ = \{Jonas \mapsto Canon,\ Vaidas \mapsto Nikon,$
  $\quad Vaiva \mapsto Sony,\ Jonas \mapsto Sony,\ Sandra \mapsto Pentax\}$
  its inverse is
  $owns\_camera^{-1} \ = \{Canon \mapsto Jonas,\ Nikon \mapsto Vaidas,$
  $\quad Sony \mapsto Vaiva,\ Sony \mapsto Jonas,\ Pentax \mapsto Sandra\}$
- Relational image $R[S]$. Returns a subset of the relation range related to any element from the given set S
- Example: for the relation *owns_camera* defined above
  $owns\_camera[\{Jonas, Vaiva\}] \ = \ \{Canon,\ Sony\}$

More operations on relations (cont.)

- Relational composition $R_1; R_2$.
  Composition of relations $R_1 \in S \leftrightarrow T$ and $R_2 \in T \leftrightarrow U$ is relation
  $\{(s \mapsto u) \mid \exists t.\ (s \mapsto t) \in R_1 \land (t \mapsto u) \in R_2\}$

- Example: Suppose we are given a relation between camera models and their brands $cmodels \in CMODEL \leftrightarrow CAMERA$,
  for instance,
  $cmodels \;=\; \{350 \mapsto Canon, 60D \mapsto Canon, D5200 \mapsto Nikon, \ldots\}$

  Then the previously defined *owns_camera* can be built as a result of a composition of the relations $owns\_cmodel \in PERSON \leftrightarrow CMODEL$ and $cmodels \in CMODEL \leftrightarrow CAMERA$, i.e.,

  $owns\_camera \;=\; owns\_cmodel \,;\, cmodels$

More operations on relations (cont.)

- Relational overriding $R_1 \Leftarrow\!\!+ R_2$ (ascii R_1 <+ R_2).

  Updating the relation R_1 by R_2:
  $R_1 \Leftarrow\!\!+ R_2 \;=\; (dom(R_2) \lhd\!\!\!- R_1) \cup R_2$

- Example:
  $owns\_camera \Leftarrow\!\!+ \{Jonas \mapsto Nikon\}$

  Two pairs $Jonas \mapsto Canon$ and $Jonas \mapsto Sony$ are removed, one pair $Jonas \mapsto Nikon$ is added

More operations on relations (cont.)

- Identity relation: $id \in S \leftrightarrow S$

  The relation contains pairs $(s \mapsto s)$ for all $s \in S$
  (each element is only related to itself)

- Example: suppose we have the relation

  $$connected \in CITY \leftrightarrow CITY$$

  containing all the road connections between cities
  (encoding a graph representation of roads between cities)

  Then the requirements "Roads can only be between different cities" can be formulated as

  $$connected \cap id = \varnothing$$

Printer access example: requirements

1. The system purpose is to manage user access to printers
2. The dynamic information about which user has currently access to to which printer is stored by the system
3. This information can be updated by three operations: granting access, blocking access, or banning the user
4. In the first case, the user can be given access to a specific printer
5. In the second case, the user can be blocked from accessing a specific printer
6. In the last case, the user can be banned from using any printer

Printer access example: requirements

7. Each printer supports a number of printing options (like color or double side printing)
8. All printers support at least one printing option
9. Also, all options are supported by at least one printer
10. The information about all printer options is static and known beforehand
11. The user can send an enquiry about whether he or she has a possibility to use a specific printing option
12. The manager can send an enquiry about by all the users of a specific printer

Printer access example: context

CONTEXT

Access_ctx

SETS

$USER$

$PRINTER$

$OPTION$

$PERMISSION$

CONSTANTS

$Options, Ok, No\_permission$

AXIOMS

$Options \in PRINTER \leftrightarrow OPTION$

$dom(Options) = PRINTER$

$ran(Options) = OPTION$

$partition(\{PERMISSION, \{Ok\}, \{NoAccess\})$

END

Printer access example: machine

MACHINE

Access_mch

SEES

Access_ctx

VARIABLES

$access, last\_query, pr\_users$

INVARIANT

$access \in USER \leftrightarrow PRINTER$

$last\_query \in PERMISSION$

$pr\_users \in \mathbb{P}(USERS)$

INITIALISATION

$access, pr\_users := \varnothing, \varnothing$

$last\_query := NoAccess$

Printer access example: machine

EVENTS

$add\_access \ =$

ANY u, p

WHEN $u \in USER \wedge p \in PRINTER$

THEN $access := access \cup \{u \mapsto p\}$ **END**

$block\_access \ =$

ANY u, p

WHEN $u \in USER \wedge p \in PRINTER \wedge (u \mapsto p) \in access$

THEN $access := access \setminus \{u \mapsto p\}$ **END**

$ban\_user \ =$

ANY u

WHEN $u \in USER \wedge u \in dom(access)$

THEN $access := \{u\} \lhd access$ **END**

...

Printer access example: machine

...

$perm\_query\_OK \; =$

$\quad$ **ANY** u, o

$\quad$ **WHEN** $u \in USER \wedge o \in OPTION$

$\qquad\qquad (u \mapsto o) \in access; Options$

$\quad$ **THEN** $perm\_query := Ok$ **END**

$perm\_query\_NOK \; =$

$\quad$ **ANY** u, o

$\quad$ **WHEN** $u \in USER \wedge o \in OPTION$

$\qquad\qquad (u \mapsto o) \notin access; Options$

$\quad$ **THEN** $perm\_query := NoAccess$ **END**

...

Printer access example: machine

...

$printer\_query =$

ANY p

WHEN $p \in PRINTER$

THEN $pr\_users := access^{\sim}[\{p\}]$

END

END

Functions

- Functions form a special class of relations that satisfy additional requirement: any element of the source set can be related to no more than 1 element of the target
- Functionality requirement mathematically:

$$\forall x, y, z.\ \ (x \mapsto y) \in R\ \wedge\ (x \mapsto z) \in R\ \Rightarrow\ y = z$$

- Any operation applicable to a relation or a set is also applicable to a function. For example, we can talk about the domain and the range of a function or a function as a set of pairs
- If f is a function, then $f(x)$ is the result of the function f for the argument x

Total and partial functions

- Functions are called *total* if their domain is the whole source set
  Syntax: $f \in S \rightarrow T$ (or ascii $f : S$ -->T)
  where $dom(f) = S$ and $ran(f) \subseteq T$

- Example: the camera model relation is actually a total function
  $cmodels \in CMODEL \rightarrow CAMERA$
  (any camera model identifier is uniquely associated with its brand)

- Functions are called *partial* if their domain is a subset of the source set
  Syntax: $f \in S \nrightarrow T$ (or ascii $f : S$ +->T)
  where $dom(f) \subseteq S$ and $ran(f) \subseteq T$

- Example: room reservation relation is actually a partial function
  $reserved \in ROOM \nrightarrow CUSTOMER$
  (each reserved room has the unique customer that served it, however, not all rooms must be reserved)

Arrays

- An array is a named, indexed collection of values of a given type.
- The array values can be accessed (read and updated) by using appropriate indexes.
- If we use $1..n$ (for some $n \in \mathbb{N}$) as our index set, then an array (containing elements of type S) can be modelled as a function from $1..n$ to S.
- In fact, any set can be used as the index set for arrays. Therefore, arrays can be usually modelled as total functions from S (index set) to T (the type of array values).

Functional (array) assignment

- The notation used to describe machine actions (i.e., assignments in the machine events) allows us to directly assign values to indexed elements of arrays:

$$a(i) \ := \ E$$

- This is just syntactic sugaring for the following assignment:

$$a \ := \ a \Leftarrow \{(i \mapsto E)\}$$

- The assignment also works if a is modelled as a partial function. However, if we want to check/read values from such an array, we have to ensure/prove (by using the event guards and/or machine invariants) that the used index belongs to the function domain, i.e., $i \in dom(a)$

- $reserved(r) \ := \ FALSE$ **OK!**
- $nItems(j) \ := \ nItems(i) + 1$ **only if** $i \in dom(nItems)$

Vending machine example revisited

New (additional) requirements:

11. The system stores the constant information about the prices of served items
12. The payment operation is successful only if the supplied payment (credit) is sufficient to cover the price of the chosen item
13. The amount of the served items in the vending machine is limited
14. The system stores the current number of available items (for each item)

Vending machine example revisited (cont.)

15. The item selection operation is only possible if the number of available numbers is positive
16. After serving the product, the system decreases the corresponding number of available items of this kind
17. Initially, the system contains the pre-defined number of each item
18. The number of available items can be increased by loading some number of items of particular kind

Vending machine example: context

CONTEXT Vending_ctx
SETS
 CHOICES
CONSTANTS
 None
 price
 initial_amount
 Items
AXIOMS
 $None \in CHOICES$
 $price \in CHOICES \nrightarrow \mathbb{N}1$
 $dom(price) = CHOICES \setminus \{None\}$
 $initial\_amount \in \mathbb{N}1$
 $Items \subseteq CHOICES$
 $Items = CHOICES \setminus \{None\}$
END

Vending machine example: machine

MACHINE

Vending_mch

SEES

Vending_ctx

VARIABLES

ready, choice, payed, served, nItems

INVARIANT

...

$nItems \in Items \rightarrow \mathbb{N}$

$choice \neq None \ \wedge \ served = FALSE \ \Rightarrow \ nItems(choice) > 0$

INITIALISATION

...

$nItems := Items \times \{initial\_amount\}$

Vending machine example: machine (cont.)

EVENTS

$\quad choose \;=$

$\qquad$ **ANY** $choice$ **WHERE** ... $nItems(choice) > 0$

$\qquad$ **THEN** ... **END**

$\quad cancel \;=$ ...

$\quad pay \;=$ **ANY** $credit$

$\qquad$ **WHERE** ... $credit > price(choice)$

$\qquad$ **THEN** ... **END**

$\quad serve \;=$

$\qquad$ **WHEN** ...

$\qquad$ **THEN** ... $nItems(choice) := nItems(choice) - 1$ **END**

...

Vending machine example: machine (cont.)

Additional event for refilling items:

...

$add\_items \ =$

ANY $it, \ n$

WHERE

$ready = TRUE$

$it \in Items$

$n \in \mathbb{N}1$

THEN

$nItems(it) := nItems(it) + n$

END