11. The map types

The outline of this part:

- What is a map?
- The type constructor
- Operators
- Specification using map
11.1 What is a map?

A map is a finite set of pairs, describing a mapping between two sets. It is a special function.
A map (or sometimes we call it "map value") is represented with a notation similar to the set notation:

\[ \{a_1 \rightarrow b_1, a_2 \rightarrow b_2, ..., a_n \rightarrow b_n\} \]

Each \(a_i \rightarrow b_i\) (\(i=1..n\)) denotes a pair which is known as maplet.

For example, the map illustrated in the Figure above is given as follows:

\[ \{a \rightarrow 2, b \rightarrow 2, c \rightarrow 3, d \rightarrow 1\} \]

An empty map is expressed as:

\[ \{\rightarrow\} \]
Important property:

A map usually describes a many-to-one mapping: it allows the mapping from many elements in the domain to the same element in the range, but does not allow the mapping from the same element in the domain to different elements in the range.
11.2 The type constructor

A map type is declared using the map type constructor:

\[ \text{map} \ldots \text{to} \ldots \]

Example:

\[ T = \text{map} \ T_1 \text{ to } T_2 \]

declares a map type \( T \) with the domain type \( T_1 \) and and the range type \( T_2 \). In other words, \( T \) contains all the possible maps that associate values in \( T_1 \) with the values in \( T_2 \).
Another concrete example:

\[
A = \text{map nat to char}
\]

declarations a map type \(A\) whose domain type is \text{nat} and range type is \text{char}.

Examples: possible map values of type \(A\):

\[
\{1 \rightarrow 'a', 2 \rightarrow 'b', 3 \rightarrow 'c', 4 \rightarrow 'd'\}
\]
\[
\{5 \rightarrow 'u', 15 \rightarrow 'v', 25 \rightarrow 'w'\}
\]
\[
\{10 \rightarrow 'x', 20 \rightarrow 'y'\}
\]
\[
\{50 \rightarrow 'r'\}
\]
\[
\{-->\}
\]
Note that: **domain type** and **range type** of a map type can be an infinite set, although **a concrete map value** derived from the map type must contain only **finite maplets** (elements of a map).
11.3 Operators

(1) Constructors
Two constructors: map enumeration and map comprehension.

(1.1) Map enumeration

The general format:
   \{a_1 --> b_1, a_2 --> b_2, ..., a_n --> b_n\}

Examples:
   \{3 --> 'a', 8 --> 'b', 10 --> 'c'\}
   \{"Hosei University" --> "Japan", "University of Manchester" --> "U.K.", "Jiaotong University" --> "China"\}
   \{1 --> s(1), 2 --> s(2), 3 --> s(3)\}
(1.2) Map comprehension

\{a \rightarrow b \mid a: T1, b: T2 & P(a, b)\} \text{ or } \{a \rightarrow b \mid P(a, b)\}

Example:
\{x \rightarrow y \mid x: \{5, 10, 15\}, y: \{10, 20, 30\} & y = 2 \times x\} =
\{5 \rightarrow 10, 10 \rightarrow 20, 15 \rightarrow 30\}

defines a qualified map, but the following map comprehension defines an illegal map:

\{x \rightarrow y \mid x: \{1, 2, 3\}, y: \{5, 10, 15, 20\} & y > x \times 5\} =
\{1 \rightarrow 10, 1 \rightarrow 15, 1 \rightarrow 20, 2 \rightarrow 15, 2 \rightarrow 20, 3 \rightarrow 20\}

Why?
(2) Other operators

(2.1) Map application

Let m be a map:

\[ m: \text{map } T_1 \text{ to } T_2; \]

Then m can be applied to an element in its domain to yield an element in its range.

Example:

\[ m(a) \]

denotes an application to element a in its domain.
Example: let

\[ m_1 = \{5 \rightarrow 10, 10 \rightarrow 20, 15 \rightarrow 30\} \]

Then

\[ m_1(5) = 10 \]
\[ m_1(10) = 20 \]
\[ m_1(15) = 30 \]

Note that when \( m_1 \) applies to number 2, the result of the application is undefined:

\[ m_1(2) = \text{undefined.} \]
(2.2) Domain and range (\textit{dom}, \textit{rng})

Let \( m \) be a map:

\[
m: \text{map } T_1 \text{ to } T_2;
\]

Then the domain of \( m \) is a subset of \( T_1 \) and its range is a subset of \( T_2 \), which can be obtained by applying the operators \textit{dom} and \textit{rng}, respectively.

\[
\text{dom}: \text{map } T_1 \text{ to } T_2 \rightarrow \text{set of } T_1
\]

Example:

let \( m_1 = \{5 \rightarrow 10, 10 \rightarrow 20, 15 \rightarrow 30\} \)

Then

\[
\text{dom}(m_1) = \{5, 10, 15\}
\]
The range operator \texttt{rng} yields, when applied to a map, the set of the second elements of all the maplets in the map.

\texttt{rng}: \text{map } T1 \text{ to } T2 \rightarrow \text{set of } T2
\texttt{rng}(m) = \{m(a) \mid a \text{ inset } \text{dom}(m)\}

Example:
\texttt{rng}(m1) = \{10, 20, 30\}
(2.3) Domain and range restriction to \((\text{domrt, rngrt})\)

Given a map and a set, sometimes we may want to obtain the subset of the map whose domain or range is restricted to the set. Such operations are known as domain restriction to and range restriction to, respectively.

\[
\text{domrt}: \text{set of } T1 \times \text{map } T1 \to T2 \to \text{map } T1 \to T2 \\
\text{domrt}(s, m) = \{ a \to m(a) | a \text{ inset } \text{inter}(s, \text{dom}(m)) \}
\]

\[
\text{rngrt}: \text{map } T1 \to T2 \times \text{set of } T2 \to \text{map } T1 \to T2 \\
\text{rngrt}(m, s) = \{ a \to m(a) | m(a) \text{ inset } \text{inter}(s, \text{rng}(m)) \}
\]
Examples: let

\[ m_1 = \{5 \rightarrow 10, 10 \rightarrow 20, 15 \rightarrow 30\} \]
\[ s_1 = \{5, 10\}. \]

Then

\[ \text{domrt}(s_1, m_1) = \{5 \rightarrow 10, 10 \rightarrow 20\} \]
\[ \text{rngrt}(m_1, s_1) = \{5 \rightarrow 10\} \]
In contrast to "domain restriction to" and "range restriction to" operations, sometimes we may want to derive a subset of a map whose domain or range is the subset of the domain or range of the map that is disjoint with a given set. Such operations are called domain restriction by and range restriction by, respectively.

\[
\text{domrb}: \text{set of T1} \times \text{map T1 to T2} \rightarrow \text{map T1 to T2} \\
\text{domrb}(s, m) == \{ a \rightarrow m(a) \mid a \text{ inset } \text{diff}(\text{dom}(m), s) \}
\]

\[
\text{rngrb}: \text{map T1 to T2} \times \text{set of T2} \rightarrow \text{map T1 to T2} \\
\text{rngrb}(m, s) == \{ a \rightarrow m(a) \mid m(a) \text{ inset } \text{diff}(\text{rng}(m), s) \}
\]
Examples:

\[ \text{domrb}(s1, m1) = \{15 \rightarrow 30\}\]
\[ \text{rngrb}(m1, s1) = \{10 \rightarrow 20, 15 \rightarrow 30\}\]
(2.5) Override (override)

Overriding is an operation of performing a union of two maps \( m_1 \) and \( m_2 \), denoted by \( \text{override}(m_1, m_2) \), with the restriction: if a maplet in map \( m_2 \) shares the first element with a maplet in \( m_1 \), the resulting map only includes the maplet in \( m_2 \) as its element.

\[
\text{override}: \text{map } T_1 \text{ to } T_2 \times \text{map } T_1 \text{ to } T_2 \rightarrow \text{map } T_1 \text{ to } T_2 \\
\text{override}(m_1, m_2) \equiv \{ a \rightarrow b \mid \\
a: \text{union}(\text{dom}(m_1), \text{dom}(m_2)) \land \\
a \text{ inset } \text{dom}(m_2) \Rightarrow b = m_2(a) \text{ and } \\
a \text{ notin } \text{dom}(m_2) \Rightarrow b = m_1(a) \}
\]
Example: let 
\[ m_1 = \{5 \rightarrow 10, \ 10 \rightarrow 20, \ 15 \rightarrow 30\}, \]
\[ m_2 = \{10 \rightarrow 5, \ 15 \rightarrow 50, \ 4 \rightarrow 20\}. \]

Then
\[
\text{override}(m_1, m_2) = \\
\{10 \rightarrow 5, \ 15 \rightarrow 50, \ 4 \rightarrow 20, \ 5 \rightarrow 10\}
\]

Notice: \text{override} is not commutative, that is,
\[
\text{override}(m_1, m_2) \not\equiv \text{override}(m_2, m_1)
\]
holds in general.

Example: compare \text{override}(m_1, m_2) to the following:
\[
\text{override}(m_2, m_1) = \{5 \rightarrow 10, \ 10 \rightarrow 20, \\
15 \rightarrow 30, \ 4 \rightarrow 20\} \]
(2.6) Map inverse (inverse)
Map inverse is an operation that yields a map from a given map by exchanging the first and second elements of every maplet of the given map.

\[
\text{inverse: map } T_1 \text{ to } T_2 \rightarrow \text{ map } T_2 \text{ to } T_1
\]
\[
\text{inverse}(m) == \{a \rightarrow b \mid a: \text{rng}(m), b: \text{dom}(m) \& a = m(b)\}
\]
Example: let $m1 = \{5 \rightarrow 10, \ 8 \rightarrow 20, \ 2 \rightarrow 30\}$
Then

\[
\text{inverse}(m1) = \{10 \rightarrow 5, \ 20 \rightarrow 8, \ 30 \rightarrow 2\}
\]

However, if the map defines a \textbf{many-to-one} rather than \textbf{one-to-one} mapping between its domain and range, application of the \textbf{inverse} operator is \textbf{undefined}.
(2.7) Map composition (\texttt{comp})

Map composition is an operation that forms a more complicated map from two maps.

\[
\text{comp} : \text{map } T_1 \text{ to } T_2 \times \text{map } T_2 \text{ to } T_3 \rightarrow \text{map } T_1 \text{ to } T_3
\]

\[
\text{comp}(m_1, m_2) = \{a \rightarrow b \mid a : \text{dom}(m_1), b : \text{rng}(m_2) \land \\
\exists [x : \text{rng}(m_1)] \mid x \text{ inset } \text{dom}(m_2) \land x = m_1(a) \land b = m_2(x)\}
\]
Example: let

\[ m_1 = \{5 \rightarrow 10, 8 \rightarrow 20, 2 \rightarrow 4\}, \]
\[ m_2 = \{10 \rightarrow 5, 15 \rightarrow 5, 4 \rightarrow 20\}, \]

Then the composition of \( m_1 \) and \( m_2 \) is:

\[ \text{comp}(m_1, m_2) = \{5 \rightarrow 5, 2 \rightarrow 20\} \]
We use $m_1 = m_2$ to mean $m_1$ is identical to $m_2$, and $m_1 \not= m_2$ to mean $m_1$ is different from $m_2$. Formally,

$$m_1 = m_2 \iff \text{dom}(m_1) = \text{dom}(m_2) \text{ and } \text{rng}(m_1) = \text{rng}(m_2) \text{ and } \forall[a: \text{dom}(m_1), b: \text{rng}(m_1)] | b = m_1(a) \iff b = m_2(a)$$

$$m_1 \not= m_2 \iff \text{not } m_1 = m_2$$
Let us reconsider defining the type Account with a map type. Since every customer's account number is unique and it is common to allow one customer to have only one account of the same kind in the same bank, the customer account can be modeled as a map from the account number to the account data including password and balance.
Account = map AccountNumber to AccountData;

AccountNumber = nat;
AccountData = composed of
    password: nat
    balance: real
end;

We then redefine the processes Check_Password, Withdraw, and Show_Balance as follows:
process Check_Password(card_id: AccountNumber, pass: nat)
    confirm: bool
ext rd account_file: Account
post card_id inset dom(account_file) and
    account_file(card_id).password =  pass and confirm =
    true
    or
    card_id notin dom(account_file) and confirm = false
comment If the given account number card_id and password
    pass are matching with the account_file, the output confirm is
    assigned with true, otherwise, it is assigned with false.
end_process;
process Withdraw(card_id: AccountNumber, amount: real)
cash: real
ext wr account_file: Account
pre card_id inset dom(account_file) and amount <=
account_file(card_id).balance
post account_file = override(~account_file, {card_id -->
mk_AccountData(~account_file(card_id).password,
~account_file(card_id).balance - amount)}) and
cash = amount
comment The precondition requires that the provided card_id
be registered in the account_file and the requested amount
to withdraw be smaller or equal to the current balance. The
updating of the current balance of the account with the
account number card_id is expressed by a map overriding
operation: the updated balance is the result of subtracting
the requested amount from the current balance.
end_process;
process Show_Balance(card_id) bal: real
ext rd account_file: Account
pre card_id inset dom(account_file)
post bal = account_file(card_id).balance
comment The account number card_id must exist in the account_file before the execution of the process. The assignment of the current balance to the output variable bal is reflected by a map application in the postcondition.
end_process;
1. Tell the similarity and difference between a map and function.

2. Given two sets $T_1 = \{1, 2\}$, $T_2 = \{10, 11\}$, construct a map type with $T_1$ being its domain type and $T_2$ being its range type, and enumerate all the possible maplets of the map type.
Let $m_1$ and $m_2$ be two maps of the map type from $\text{nat0}$ to $\text{nat0}$; 
$m_1 = \{1 \rightarrow 10, \ 2 \rightarrow 3, \ 3 \rightarrow 30\},$
$m_2 = \{2 \rightarrow 40, \ 3 \rightarrow 1, \ 4 \rightarrow 80\}$, and $s = \{1, 3\}$. Then evaluate the expressions:

a. $\text{dom}(m_1) = ?$
b. $\text{dom}(m_2) = ?$
c. $\text{rng}(m_1) = ?$
d. $\text{rng}(m_2) = ?$
e. $\text{domrt}(s, m_1) = ?$
f. $\text{domrt}(s, m_2) = ?$
g. $\text{rngrt}(m_1, s) = ?$
h. $\text{rngrt}(m_2, s) = ?$
i. domrb(s, m1) = ?
j. domrb(s, m2) = ?
k. rngrb(m1, s) = ?
l. rngrb(m2, s) = ?
m. override(m1, m2) = ?
n. override(m2, m1) = ?
o. inverse(m1) = ?
p. inverse(m2) = ?
q. comp(m1, m2) = ?
r. comp(m2, m1) = ?
s. m1 = m2 <=> ?
t. m1 <> m2 <=> ?
4. Give a concrete example to explain that 
   \( \text{comp}(m_1, m_2) \) is defined whereas \( \text{comp}(m_2, m_1) \) is 
   undefined.

5. Define \textit{BirthdayBook} as a map type from the type \textit{Person} to the type \textit{Birthday}, and specify the 
   processes: \textit{Register}, \textit{Find}, \textit{Delete}, and \textit{Update}. All 
   the processes access or update the external 
   variable \textit{birthday\_book} of the type \textit{BirthdayBook}. 
   The process \textit{Register} adds a person's birthday to \textit{birthday\_book}; \textit{Find} detects the birthday for a 
   person in \textit{birthday\_book}; \textit{Delete} eliminates the 
   birthday for a person from \textit{birthday\_book}; and \textit{Update} replaces the wrong birthday existing in 
   \textit{birthday\_book} with a correct birthday.