8. The set types

The set types are one of the compound types available in SOFL, and usually used for the abstraction of data items that have a collection of elements.

The outline of this part:
- What is a set?
- Set type constructors
- Constructors and operators on sets
- Specification with set types
What is a set?

A set is an **unordered** collection of **distinct** objects where each object is known as an element of the set. In SOFL, we only allow finite sets, that is, a set contains finite number of elements.

For example:

1. A class is a set of students.
2. A car park is a set of cars.

A set value is marked by braces, thus we have

1. \{5, 9, 10\}
2. \{"John", "Chris", "David", "Jeff"\}
3. \{"Java", "Pascal", "C", "C++", "Fortran"\}

**Notice:** \{a, a, b\} is not a legal set.
Set type declaration

A set type is declared by applying the set type constructor to an element type.

Let $T$ be an arbitrary type, $A$ be a set type to be defined. Then the definition of $A$ has the form:

$$A = \text{set of } T$$

where \text{set of} is called “set type constructor”.

Formally, $A$ is a power set of $T$:

$$A = \{x \mid \text{subset}(x, T)\}$$

Where \text{subset}(x, T) means that $x$ is a subset of $T$. 
For example: let $A$ be defined as follows:

type
$A = $ set of $\{<$DOG$>, <$CAT$>, <$COW$>$\}

This means:

$A = \{\{\}\}, \{<$DOG$>\}, \{<$CAT$>\}, \{<$COW$>\},$
\{<$DOG$>, <$CAT$>\}, \{<$DOG$>, <$COW$>\},
\{<$CAT$>, <$COW$>\}, \{<$DOG$>, <$CAT$>, <$COW$>\}$
Set variable declaration:

Let $s$ be a variable of type $A$, which is defined as:

```plaintext
var
s: A;
```

then $s$ can take any of the values available in $A$, thus:

- $s = \{ \} \text{(empty set) or}$
- $s = \{<\text{DOG}>\}$ or
- $s = \{<\text{CAT}>\}$ or
- $s = \{<\text{COW}>\}$ or
- $s = \{<\text{DOG}, <\text{CAT}>\}$ or
- $s = \{<\text{DOG}, <\text{COW}>\}$ or
- $s = \{<\text{CAT}, <\text{COW}>\}$ or
- $s = \{<\text{DOG}, <\text{CAT}, <\text{COW}>\}$
Constructors and operators on sets

1. Constructors
A constructor of set types is a special operator that constitutes a set value from the elements of an element type.

There are two set constructors:
set enumeration and set comprehension.
A set enumeration has the format:

\{e_{1}, e_{2}, ..., e_{n}\}

where e_{i} (i=1..n) are the elements of the set \{e_{1}, e_{2}, ..., e_{n}\}.

Examples:

\{5, 9, 10, 50\}
\{‘a’, ‘t’, ‘l’\}
A set comprehension has the form:

\[ \{ e(x_1, x_2, \ldots, x_n) \mid x_1: T_1, x_2: T_2, \ldots, x_n: T_n \& P(x_1, x_2, \ldots, x_n) \} \]

where \( n \geq 1 \).

The set comprehension defines a collection of values resulting from evaluating the expression \( e(x_1, x_2, \ldots, x_n) \) (\( n \geq 1 \)) under the condition that the involved variables \( x_1, x_2, \ldots, x_n \) take values from sets (or types) \( T_1, T_2, \ldots, T_n \), respectively, and satisfies property \( P(x_1, x_2, \ldots, x_n) \).
Examples:

\{x \mid x: \text{nat} \& 1 < x < 5\} = \{2, 3, 4\}
\{y \mid y: \text{nat0} \& y \leq 5\} = \{0, 1, 2, 3, 4, 5\}
\{x + y \mid x: \text{nat0}, y: \text{nat0} \& 1 < x + y < 8\} =
\{2, 3, 4, 5, 6, 7\}
\{i \mid i: \text{nat0} \& 9 < i < 4\} = \{\}\n
We can also use the following special notation to represent a set containing an interval of integers:
\{i, \ldots, k\} = \{j \mid j: \text{int} \& i \leq j \leq k\}
Thus:
\{1, \ldots, 5\} = \{1, 2, 3, 4, 5\}
\{-2, \ldots, 2\} = \{-2, -1, 0, 1, 2\}
2. Operators

2.1 Membership (inset)

inset: \( T \times \text{set of} \ T \rightarrow \text{bool} \)

Examples:

7 \( \text{inset} \) \{4, 5, 7, 9\} \( \leftrightarrow \) true
3 \( \text{inset} \) \{4, 5, 7, 9\} \( \leftrightarrow \) false
2.2 Non-membership (notin)

notin: T * set of T --> bool

Examples:
7 notin {4, 5, 7, 9} <=> false
3 notin {4, 5, 7, 9} <=> true
2.4 Cardinality (card)

\[ \text{card: set of } T \rightarrow \text{nat0} \]

Examples:

\[ \text{card}([5, 7, 9]) = 3 \]
\[ \text{card}(['h', 'o', 's', 'e', 'i']) = 5 \]
2.5 Equality and inequality (=, <>)

=: set of T * set of T --> bool
s1 = s2
== forall[x: s1] | x inset s2 and card(s1) = card(s2)

“==“ means “defined as”.

<>: set of T * set of T --> bool
s1 <> s2
== (exists[x: s1] | x notin s2) or (exists[x: s2] | x notin s1)

Examples:
   {5, 15, 25} = { 25, 15, 5} <=> true
   {5, 15, 25} <> {5, 20, 30} <=> true
2.6 Subset (subset)

subset: set of T * set of T --> bool
subset(s1, s2) == forall[x: s1] | x inset s2

Examples:
Let s1 = \{5, 15, 25\}, s2 = \{5, 10, 15, 20, 25, 30\}. Then:

subset(s1, s2) <=> true    subset(s2, s1) <=> false
subset(\{ \}, s1) <=> true    subset(s1, s1) <=> true
2.7 Proper subset (psubset)

\[
\text{psubset: set of } T \times \text{ set of } T \rightarrow \text{ bool} \\
\text{psubset}(s1, s2) == \text{ subset}(s1, s2) \text{ and } s1 <> s2
\]

Examples:
let \( s1 = \{5, 15, 25\} \) and \( s2 = \{5, 10, 15, 25, 30\} \).
Then:
\[
\text{psubset}(s1, s2) <=> \text{ true} \\
\text{psubset}(s1, s1) <=> \text{ false} \\
\text{psubset}(s2, s1) <=> \text{ false} \\
\text{psubset}({ }, s1) <=> \text{ true}
\]
2.8 Member access (get)

get: set of T --> T
get(s) == if s <> { } then x else nil
Where x inset s.

Examples: assume s = {5, 15, 25}, then
get(s) = 5 or
get(s) = 15 or
get(s) = 25
And s still remains the same as before:
s = {5, 15, 25}. 
2.9 Union (union)

union: set of T * set of T --> set of T
union(s1, s2) == {x | x inset s1 or x inset s2}

Examples:
union({5, 15, 25}, {15, 20, 25, 30}) = {5, 15, 25, 20, 30}
union({15, 20, 25, 30}, {5, 15, 25}) = {5, 15, 25, 20, 30}
The union operator is commutative. Thus, 
union(s1, s2) = union(s2, s1).

It is also associative, that is, 
union(s1, union(s2, s3)) = 
union(union(s1, s2), s3).

Due to these properties, the operator union can be extended to deal with more than two sets:  
union: set of T * set of T * ... * set of T --> set of T  
union(s1, s2, ..., sn) == \{x | x \in s1 \text{ or } x \in s2 \text{ or } ... \text{ or } x \in sn\}
2.10 Intersection (inter)

inter: set of T * set of T --> set of T
inter(s1, s2) == \{x \mid x \text{ inset } s1 \text{ and } x \text{ inset } s2\}

For example, let s1 = \{5, 7, 9\},
\hspace{1cm} s2 = \{7, 10, 9, 15\},
\hspace{1cm} s3 = \{8, 5, 20\}.

Then
\hspace{1cm} inter(s1, s2) = \{7, 9\}
\hspace{1cm} inter(s1, s3) = \{5\}
\hspace{1cm} inter(s2, s3) = \{\}
The \textit{inter} operator is commutative and associative. That is,

\begin{align*}
\text{inter}(s_1, s_2) &= \text{inter}(s_2, s_1), \\
\text{inter}(s_1, \text{inter}(s_2, s_3)) &= \text{inter}(\text{inter}(s_1, s_2), s_3).
\end{align*}

We can also extend the \textit{inter} operator to deal with more than two operands:

\begin{align*}
\text{inter}: \text{set of } T \times \text{set of } T \times \ldots \times \text{set of } T &\rightarrow \text{set of } T \\
\text{inter}(s_1, s_2, \ldots, s_n) &= \{x \mid x \text{ inset } s_1 \text{ and } x \text{ inset } s_2 \text{ and } \ldots \text{ and } x \text{ inset } s_n\}.
\end{align*}
2.11 Difference (diff)

diff: set of T * set of T --> set of T

\[ \text{diff}(s1, s2) = \{ x \mid x \text{ inset } s1 \text{ and } x \text{ notin } s2 \} \]

For example, let \( s1 = \{5, 7, 9\} \)
\( s2 = \{7, 10, 9, 15\} \)
\( s3 = \{8, 12\} \).

Then

\[ \text{diff}(s1, s2) = \{5\} \]
\[ \text{diff}(s1, s3) = \{5, 7, 9\} \]
\[ \text{diff}(s2, s1) = \{10, 15\} \]
\[ \text{diff}(s1, \{\}) = s1 \]
2.12 Distributed union (\textit{dunion})

A set can be a set of sets, and the distributed union of such a set is an operation that obtains the union of all the member sets of the set.

\textbf{dunion}: set of set of T $\to$ set of T \\
dunion(s) == union(s1, s2, ..., sn)

where $s = \{s_1, s_2, ..., s_n\}$.

Examples:

Let $s_1 = \{\{5, 10, 15\}, \{5, 10, 15, 25\}, \{10, 25, 35\}\}$

Then

\textbf{dunion}(s1) = \textbf{union}(\{5, 10, 15\}, \{5, 10, 15, 25\}, \{10, 25, 35\})

$= \{5, 10, 15, 25, 35\}$
2.13 Distributed intersection (dinter)

dinter: set of set of T --> set of T

dinter(s) == inter(s₁, s₂, ..., sₙ)

where s = {s₁, s₂, ..., sₙ}.

For example, let

s = {{5, 10, 15}, {5, 10, 15, 25}, {10, 25, 35}}.

Then

\[ \text{dinter}(s) = \text{inter}({{5, 10, 15}, {5, 10, 15, 25}, {10, 25, 35}}) = \{10\} \]
2.14 Power set (power)
Given a set, we can apply the operator power to yield its power set that contains all the subsets of the set, including the empty set.

\[
power : \text{set of } T \rightarrow \text{set of set of } T
\]
\[
power(s) = \{ s1 \mid \text{subset}(s1, s) \}
\]

Example: let \( s = \{5, 15, 25\} \). Then
\[
power(s) = \{\{\}, \{5\}, \{15\}, \{25\}, \{5, 15\}, \{15, 25\}, \{5, 25\}, \{5, 15, 25\}\}
\]
Specification with set types

An Email_Address_Book

module Email_Address_Book;
type
  Email = given;
var
  email_book: set of Email;
behav: CDFD_1;
Figure 1
process Find(e: Email) r: bool
ext rd email_book
post r = (e inset email_book)
end_process;

process Add(e: Email)
ext wr email_book
post email_book = union(~email_book, {e})
end_process;

process Delete(e: Email)
ext wr email_book
post email_book = diff(~email_book, {e})
end_process;
end_module;
Exercise 8

1. Given a set $T = \{5, 8, 9\}$, define a set type based on $T$, and list all the possible set values in the type.

2. Let $T = \{5, 8, 9\}$. Then evaluate the set comprehensions:
   a. $\{x \mid x: \text{nat} \& x < 8\}$
   b. $\{y \mid y: \text{nat0} \& y \leq 3\}$
   c. $\{x + y \mid x: \text{int}, y: \text{int} \& -2 < x + y < 3\}$
   d. $\{i \mid i: \text{set of } T \& \text{card}(i) < 3 \text{ and } \forall [x, y: i] \mid x + y \leq 13\}$
3. Let $s_1 = \{5, 15, 25\}$, $s_2 = \{15, 30, 50\}$, $s_3 = \{30, 2, 8\}$, and $s = \{s_1, s_2, s_3\}$. Evaluate the expressions:

a. $\text{card}(s_1)$

b. $\text{card}(s)$

c. $\text{union}(s_1, s_2)$

d. $\text{diff}(s_2, s_3)$

e. $\text{inter}(\text{union}(s_2, s_3), s_1)$

f. $\text{dunion}(s)$

g. $\text{dinter}(s)$

h. $\text{inter}(\text{union}(s_1, s_3), \text{diff}(s_2, \text{union}(s_1, s_3)))$
4. Write set comprehensions for the sets:
   a. a set of natural numbers whose elements are all less than 10.
   b. a set of integers whose elements are all greater than 0 and less than 10 and cannot be divided by 3.
   c. a set of prime numbers.

5. Construct a module to model a telephone book containing a set of telephone numbers. The necessary processes are Add, Find, Delete, and Update. The process Add adds a new telephone number to the book; Find tells whether a given telephone number is available or not in the book; Delete eliminates a given telephone number from the book; and Update replaces an old telephone number with a new number in the book.
6. Write a specification for a process **Merge**. The process takes two groups of students, and merge them into one group. Since the merged group will be taught by a different professor, the students from both groups may drop from the merged group (but exactly which students will drop is unknown).