

Modelling using Sequences

- Sequences
 - The finite sequence constructor
 - Value definitions: enumeration, subsequence
 - Operators on Sequences
- Case Study: the BASE Trusted Gateway

The finite sequence type constructor

In VDM-SL, a **sequence** is a finite ordered collection of values. The presence of duplicates and the order in which elements are presented is significant.

The finite sequence type constructor is:

```
seq of X
```

where x is an arbitrary type. The type `seq of X` is the class of all possible finite sequences of values drawn from the type x .

For example:

```
seq of nat1
```

```
seq of (seq of char)
```

Finite sequence value definitions

Sequence values can be represented in various ways:

- **Enumeration**, e.g. `[3, 5, 2, 5, 45]`
`[{34}, {34,7}, "Fred"]`
empty sequence `[]`

- Sequences of characters may be given as strings in quotation marks, e.g.

`['l', 'i', 'n', 'u', 'x'] = "linux"`

- **Subsequence**: If we have a sequence q then we can take an extract from q , e.g. $q(3, \dots, 5) = [q(3), q(4), q(5)]$

- **Comprehension**: The sequence comprehension notation is not often used and is described in the text.

- Note that sequences, like sets, are *finite*.

Operators on finite sequences

hd: seq of X \rightarrow X

Partial operator: s $\langle \rangle$ []
First element

tl: seq of X \rightarrow seq of X

Partial operator: s $\langle \rangle$ []
Tail (NB: a sequence!)

len: seq of X \rightarrow nat

length of sequence

elems: seq of X \rightarrow set of X

elements in the sequence
(reduced to a set)

inds: seq of X \rightarrow set of nat

indices of the sequence
{1,...,len s}

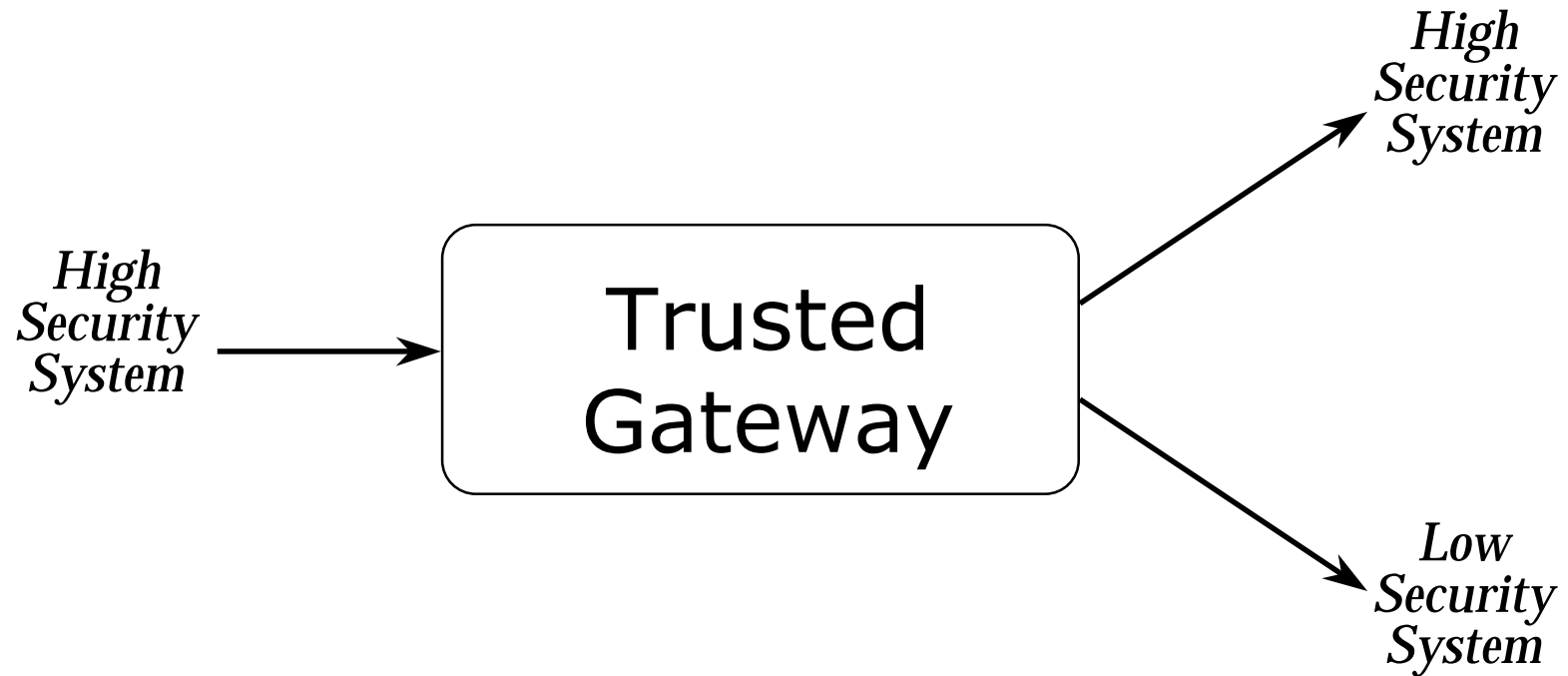
_ ^ _ : seq of X * seq of X \rightarrow seq of X

sequence concatenation

conc: seq of (seq of X) \rightarrow seq of X

conc s = the concatenation of all the sequences in s

The BASE Trusted Gateway



The BASE Trusted Gateway

Each message is a non-empty sequence of characters starting with the sequence "STR" and ending with the sequence "STP". The total length of a message must not exceed 10000 characters.

String = seq of char

Msg = String

inv m ==

The BASE Trusted Gateway

A message may be of high or low security.

```
Classification = <High> | <Low>
```

A message is defined as high security if it contains an occurrence of the string "SECRET". If it does not contain the string "SECRET" and it does contain the string "UNCLASSIFIED" then the message is treated as low-security. If neither string is present, then the message is classes as high-security.

```
Classify: Msg -> Classification
```

```
Classify(m) == ???
```

The BASE Trusted Gateway

We must model strings occurring as substrings of longer strings, so we use an auxiliary function for this:

```
occurs: String * String -> bool
occurs(s1,s2) ==
```

Then with the aid of this function, message classification is easy to model:

```
Classify: Msg -> Classification
Classify(m) ==
```

The BASE Trusted Gateway

```
Gateway :: input : seq of Msg
         outHi  : seq of Msg
         outLo  : seq of Msg
```

```
AnalyseInput: Gateway -> Gateway
```

```
AnalyseInput(g) ==
```

pre

Review Exercise

In the real Trusted Gateway, we did not search for particular strings in the incoming messages. Instead, the gateway contained two collections of strings called **categories**: the “high category” and the “low category”.

If the incoming string contained a string from the high category, then it was classed as high security. If the incoming string contained no high category strings and contained some low category strings, then it was classed as low security. Otherwise it was treated as high security.

Add the categories as sets of strings to our model of the gateway and modify the classification function appropriately. Record the restriction that there should be at most 20 strings in each category.