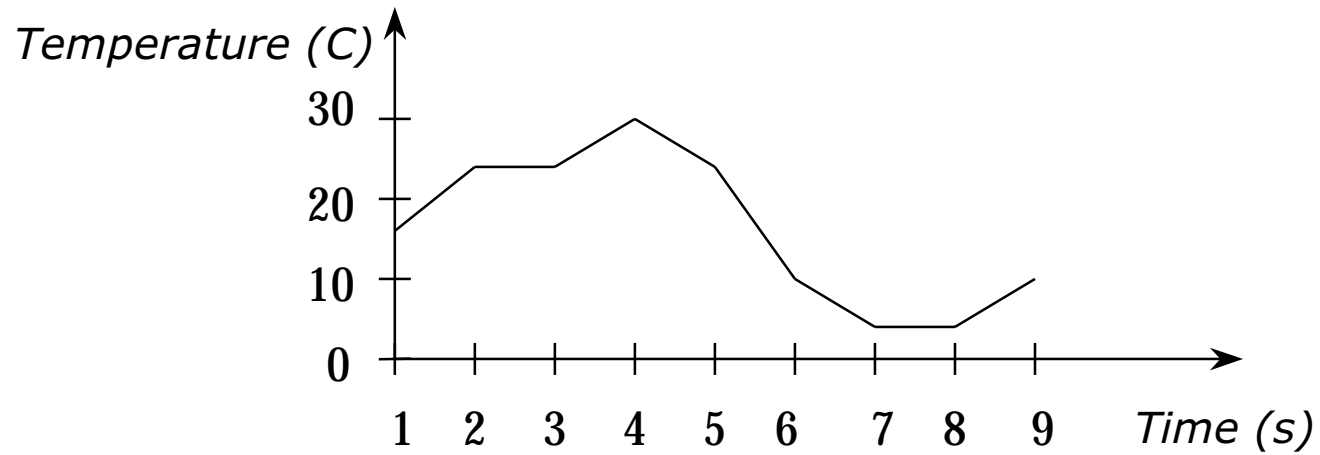


Logic

Our ability to state invariants, record preconditions and post-conditions, and the ability to reason about a formal model depend on the logic on which the modelling language is based.

- Classical logical propositions and predicates
- Connectives
- Quantifiers
- Handling undefinedness: the logic of partial functions

The temperature monitor example



The monitor records the last five temperature readings

<i>25</i>	<i>10</i>	<i>5</i>	<i>5</i>	<i>10</i>
-----------	-----------	----------	----------	-----------

The temperature monitor example

The following conditions are to be detected by the monitor:

Rising: the last reading in the sample is greater than the first

Over limit: there is a reading in the sample in excess of 400 C

Continually over limit: all the readings in the sample exceed 400 C

Safe: If readings do not exceed 400 C by the middle of the sample, the reactor is safe. If readings exceed 400 C by the middle of the sample, the reactor is still safe provided that the reading at the end of the sample is less than 400 C.

Alarm: The alarm is to be raised if and only if the reactor is not safe

Formal Model of the monitor:

Predicates

Propositions

Predicates are simply logical expressions. The simplest kind of logical predicate is a *proposition*.

A proposition is a logical assertion about a particular value or values, usually involving a Boolean operator to compare the values, e.g.

$$3 < 27$$

$$5 = 9$$

Propositions are normally either true or false (but in VDM we also have to handle undefined values - see the later notes on the Logic of Partial Functions).

Propositions have very limited value:

Predicates

General predicates

A predicate is a logical expression that is not specific to particular values but contains variables which can stand for one of a range of possible values, e.g.

$$x < 27$$

$$(x**2) + x - 6 = 0$$

The truth or falsehood of a predicate depends on the value taken by the variables.

Predicates

Predicates in the monitor example

```
Monitor :: temps : seq of int  
        alarm : bool  
  
inv m == len m.temps = 5
```

Consider a monitor m . m is a sequence so we can index into it:

First reading in m :

Last reading in m :

Predicate stating that the first reading in m is strictly less than the last reading:

The truth of the predicate depends on the value of m .

Predicates

The rising condition

The last reading in the sample is greater than the first

```
Monitor :: temps : seq of int  
        alarm : bool
```

```
inv m == len m.temps = 5
```

We can express the rising condition as a Boolean function:

```
Rising: Monitor -> bool
```

```
Rising(m) == m.temps(1) < m.temps(5)
```

For any monitor m , the expression $\text{Rising}(m)$ evaluates to true iff the last reading in the sample in m is higher than the first, e.g.

```
Rising( mk_Monitor([233,45,677,650,900], false) )
```

Basic logical operators

We build more complex logical expressions out of simple ones using logical connectives:

not negation

and conjunction

or disjunction

\Rightarrow implication (if ... then ...)

\Leftrightarrow biimplication (if and only if)

Basic logical operators

Negation

Negation allows us to state that the opposite of some logical expression is true, e.g.

The temperature in the monitor `mon` is not rising:

```
not Rising(mon)
```

Truth table for negation:

A	not A
true	false
false	true

Basic logical operators

Disjunction

Disjunction allows us to express alternatives that are not necessarily exclusive:

Over limit: *There is a reading in the sample in excess of 400 C*

```
OverLimit: Monitor -> bool
```

```
OverLimit(m) ==
```

A	B	A or B
true	true	true
true	false	true
false	true	true
false	false	false

Basic logical operators

Conjunction

Conjunction allows us to express the fact that all of a collection of facts are true.

Continually over limit: all the readings in the sample exceed 400 C

```
COverLimit: Monitor -> bool
```

```
COverLimit(m) == m.temps(1) > 400 and  
                m.temps(2) > 400 and  
                m.temps(3) > 400 and  
                m.temps(4) > 400 and  
                m.temps(5) > 400
```

A	B	A and B
true	true	true
true	false	false
false	true	false
false	false	false

Basic logical operators

Implication

Implication allows us to express facts which are only true under certain conditions ("if ... then ..."):

Safe: If readings do not exceed 400 C by the middle of the sample, the reactor is safe. If readings exceed 400 C by the middle of the sample, the reactor is still safe provided that the reading at the end of the sample is less than 400 C.

```
Safe: Monitor -> bool
```

```
Safe(m) ==
```

A	B	A ==> B
true	true	true
true	false	false
false	true	true
false	false	true

Basic logical operators

Biimplication

Biimplication allows us to express equivalence ("if and only if").

Alarm: The alarm is to be raised if and only if the reactor is not safe

This can be recorded as an invariant property:

```
Monitor :: temps : seq of int
         alarm : bool

inv m == len m.temps = 5 and
```

A	B	A \Leftrightarrow B
true	true	true
true	false	false
false	true	false
false	false	true

Quantifiers

For large collections of values, using a variable makes more sense than dealing with each case separately.

inds m.temps represents indices (1-5) of the sample

The “over limit” condition can then be expressed more economically as:

```
exists i in set inds m.temps & temps(i) > 400
```

The “continually over limit” condition can then be expressed using “forall”:

Quantifiers

Syntax:

forall *binding* & *predicate*

exists *binding* & *predicate*

There are two types of binding:

Type Binding, e.g.

x:nat

n: seq of char

*A type binding lets the bound variable range over a **type** (a possibly infinite collection of values).*

Set Binding, e.g.

i in set inds m

x in set {1,...,20}

*A set binding lets the bound variable range over a **finite set of values**.*

Quantifiers

Several variables may be bound at once by a single quantifier, e.g.

```
forall x,y in set {1,...,5} &  
    not m.temp(x) = m.temp(y)
```

Would this predicate be true for the following value of `m.temp` ?

```
[320, 220, 105, 119, 150]
```

Quantifiers

Exercises

All the readings in the sample are less than 400 and greater than 50.

Each reading in the sample is up to 10 greater than its predecessor.

There are two distinct readings in the sample which are over 400.

Quantifiers

Suppose we have to formalise the following property:

There is a "single minimum" in the sequence of readings, i.e. there is a reading which is strictly smaller than any of the other readings.

Suppose the order of the quantifiers is reversed.

Summary

- **Propositions**
- **Predicates involve free variables**
- **Predicates may be combined using connectives**
- **Free variables can range over collections of values, using quantifiers**
- **Quantifiers can be mixed**

LPF: coping with undefinedness

Suppose sensors can fail in such a way that they generate the value **Error** instead of a valid temperature.

In this case we can not make comparisons like

Error < 400

The logic in VDM is equipped with facilities for handling undefined applications of operators (e.g. if division by zero could occur).

Truth tables are extended to deal with the possibility that undefined values can occur, e.g.

A	not A
true	false
false	true
*	*

** represents the undefined value*



Disjunction in LPF

A	B	A or B
true	true	true
true	false	true
true	*	true
false	true	true
false	false	false
false	*	*
*	true	true
*	false	*
*	*	*

If one disjunct is true, we know that the whole disjunction is true, regardless of whether the other disjunct is true, false or undefined.

Conjunction in LPF

A	B	A and B
true	true	true
true	false	false
true	*	*
false	true	false
false	false	false
false	*	false
*	true	*
*	false	false
*	*	*

If one conjunct is false, we know that the whole conjunction is false, regardless of whether the other disjunct is true, false or undefined.

Implication and Biimplication in LPF

A	B	A \Rightarrow B
true	true	true
true	false	false
true	*	*
false	true	true
false	false	true
false	*	true
*	true	true
*	false	*
*	*	*

A	B	A \Leftrightarrow B
true	true	true
true	false	false
true	*	*
false	true	false
false	false	true
false	*	*
*	true	*
*	false	*
*	*	*