RSL Booleans and Logic

Overview

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

Programs versus mathematics – chaos

In *mathematics* expressions always evaluate to a value.

Example 1: 3 + 1 evaluates to 4

Example 2: 3 + 1 = 4 evaluates to true

However, *programs* may not terminate with a value.

Example:

while true do x := x + 1 **end** ; **true**

In *RSL* **chaos** represents non-terminating program constructs.

Hence

while true do x := x + 1 **end** ; **true** ≡ **chaos**

IMM/DTU, Building 322, DK–2800 Lyngby, Denmark; Phone: +45-45-257510, Fax: +45-45-930007, E-mail: ah@imm.dtu.dk February 3, 2003

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

RSL logic versus mathematical logic

Due to the possibility of non-termination, RSL Boolean value expressions are not always equivalent to:

- **true** or
- **false**

but sometimes to:

- **chaos**

Boolean operators in RSL behave like in mathematics for arguments equivalent to **true** and **false**. But we have to consider what happens, when arguments are equivalent to **chaos**.

IMM/DTU, Building 322, DK–2800 Lyngby, Denmark; Phone: +45-45-257510, Fax: +45-45-930007, E-mail: ah@imm.dtu.dk February 3, 2003

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

If Expressions

Example:

check(p2, register(p1, db)) ≡
  if p2 = p1 **then true else** check(p2, db) **end**

General form:

if *logical*-expr **then** expr1 **else** expr2 **end**

Properties:

if true then expr1 **else** expr2 **end** ≡ expr1
if false then expr1 **else** expr2 **end** ≡ expr2
if chaos then expr1 **else** expr2 **end** ≡ **chaos**

Non-strictness:

if true then expr1 **else chaos end** ≡ expr1
if false then chaos else expr2 **end** ≡ expr2

IMM/DTU, Building 322, DK–2800 Lyngby, Denmark; Phone: +45-45-257510, Fax: +45-45-930007, E-mail: ah@imm.dtu.dk February 3, 2003

Connectives

Properties:

$\sim$e $\equiv$ **if** e **then false else true**

e1 $\wedge$ e2 $\equiv$ **if** e1 **then** e2 **else false end**

e1 $\vee$ e2 $\equiv$ **if** e1 **then true else** e2 **end**

e1 $\Rightarrow$ e2 $\equiv$ **if** e1 **then** e2 **else true end**

Imply conditional logic

Truth tables

$\wedge$	**true**	**false**	**chaos**
true	**true**	**false**	**chaos**
false	**false**	**false**	**false**
chaos	**chaos**	**chaos**	**chaos**

$\vee$	**true**	**false**	**chaos**
true	**true**	**true**	**true**
false	**true**	**false**	**chaos**
chaos	**chaos**	**chaos**	**chaos**

$\Rightarrow$	**true**	**false**	**chaos**
true	**true**	**false**	**chaos**
false	**true**	**true**	**true**
chaos	**chaos**	**chaos**	**chaos**

Note:

e1 $\wedge$ e2 $\equiv$ e2 $\wedge$ e1
e1 $\vee$ e2 $\equiv$ e2 $\vee$ e1

are **not** tautologies. Why?

Quantified expressions

Examples:

$\forall$ x : **Nat** $\bullet$ (x = 0) $\vee$ (x > 0)

$\exists$ x : **Int** $\bullet$ x = 7

$\exists!$ x : **Int** $\bullet$ (x $\geq$ 0) $\wedge$ (x $\leq$ 0)

$\forall$ x : **Nat** $\bullet$ x = -7

$\forall$ x, y : **Nat** $\bullet$ ($\exists!$ z : **Nat** $\bullet$ x+y = z)

General form:

quantifier typing$_1$, ... , typing$_n$ $\bullet$ *logical*-expr

All quantification is over values in the types stated, i.e. not over **chaos**.

Equivalence and Equality Expressions

Syntactic form:

equivalence expressions: expr$_1$ $\equiv$ expr$_2$

equality expressions: expr$_1$ = expr$_2$

Context Conditions:
expr$_1$ and expr$_2$ must have same types

Use of $\equiv$ versus =

= is used *in Boolean expressions* to specify that a program should make a comparison operation.

$\equiv$ is used *in axioms* that express that two expressions should behave equivalently.

Example:

axiom
$\forall$ p1, p2 : Person, db : Database $\bullet$
check(p2, register(p1, db)) $\equiv$
if p2 = p1 **then true else** check(p2, db) **end**

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

Meaning of $\equiv$ versus $=$

$=$ and $\equiv$ differ wrt.

- undefinedness (**chaos**)
  - '$\equiv$' is two valued — the result is never **chaos**
  - '$=$' is strict, '$\equiv$' is not
  - '$\equiv$' is reflexive, '$=$' is not
- non-determinism
- side-effects (variables)
- communication

otherwise: $e_1 \equiv e_2 \simeq e_1 = e_2$

Assume e_1 and e_2 are deterministic, without side-effects and without communication, and represent distinct values.

$\equiv$	e_1	e_2	**chaos**
e_1	**true**	**false**	**false**
e_2	**false**	**true**	**false**
chaos	**false**	**false**	**true**

$=$	e_1	e_2	**chaos**
e_1	**true**	**false**	**chaos**
e_2	**false**	**true**	**chaos**
chaos	**chaos**	**chaos**	**chaos**

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

Exercise

Which of the following expressions are true?

$\exists$ i : **Nat** • i = i,
$\forall$ i : **Nat** • i = i,
$\exists$ i : **Nat** • i $\neq$ i,
$\exists$ i,j : **Nat** • i $\neq$ j,
$\forall$ i : **Nat** • $\exists$ j : **Nat** • j = i−1
$\forall$ i : **Nat** • **false**
$\exists$ i : **Nat** • **true**

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

Exercise

Express formally (in RSL) that:

1. There is at least one natural number greater than 10.
2. All pairs of natural numbers add up to 89.
3. Each consecutive pair of natural numbers add up to an odd number.
4. There is no smallest integer.
5. Between every two real numbers there is another real number.

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

Exercise

Assume given the following declarations

type Person
value
  follows_02262 : Person → **Bool**,
  has_passed_02262 : Person → **Bool**

Express formally (using the above predicates) that no person following course 02262 has already passed it.

Technical University of Denmark
Informatics and Mathematical Modelling
Computer Science and Technology

Summary

Now you should know:

- Boolean type expression
- Boolean value expressions
- special features of the logic:
  - **chaos** modelling non-termination of programs
  - evaluation of if-then-else and $\wedge$, $\vee$ and $\Rightarrow$ expressions is *conditional* (and hence non-strict) as in many programming languages
  - there is not only a "programming" equality operator $=$ (like $==$ in Java), but also an equivalence operator $\equiv$ used for specification purposes