Script **generated by TTT**

Title: Nipkow: Info2 (17.10.2014)

Date: Fri Oct 17 06:30:46 GMT 2014

Duration: 87:48 min

Pages: 19

3.6 Syntax matters

Functions are defined by one or more equations.
In the simplest case, each function is defined
by one (possibly conditional) equation:

$$\begin{array}{llll} f\ x_1\ \ldots\ x_n & & & \\ \quad \mid & test_1 & = & e_1 \\ \quad \vdots & & & \\ \quad \mid & test_n & = & e_n \end{array}$$

Each right-hand side e_i is an expression.

3.6 Syntax matters

Functions are defined by one or more equations.
In the simplest case, each function is defined
by one (possibly conditional) equation:

$$\begin{array}{llll} f\ x_1\ \ldots\ x_n & & & \\ \quad \mid & test_1 & = & e_1 \\ \quad \vdots & & & \\ \quad \mid & test_n & = & e_n \end{array}$$

Each right-hand side e_i is an expression.

3.6 Syntax matters

Functions are defined by one or more equations.
In the simplest case, each function is defined
by one (possibly conditional) equation:

$$\begin{array}{llll} f\ x_1\ \ldots\ x_n & & & \\ \quad \mid & test_1 & = & e_1 \\ \quad \vdots & & & \\ \quad \mid & test_n & = & e_n \end{array}$$

Each right-hand side e_i is an expression.

Note: `otherwise = True`

3.6 Syntax matters

Functions are defined by one or more equations.
In the simplest case, each function is defined
by one (possibly conditional) equation:

$$
\begin{array}{l}
f \;\; x_1 \;\; \ldots \;\; x_n \\
\quad \begin{array}{lll}
| \;\; test_1 & = & e_1 \\
\vdots & & \\
| \;\; test_n & = & e_n
\end{array}
\end{array}
$$

Each right-hand side e_i is an expression.
Note: `otherwise = True`

Function and parameter names must begin with a lower-case letter

3.6 Syntax matters

Functions are defined by one or more equations.
In the simplest case, each function is defined
by one (possibly conditional) equation:

$$
\begin{array}{l}
f \;\; x_1 \;\; \ldots \;\; x_n \\
\quad \begin{array}{lll}
| \;\; test_1 & = & e_1 \\
\vdots & & \\
| \;\; test_n & = & e_n
\end{array}
\end{array}
$$

Each right-hand side e_i is an expression.
Note: `otherwise = True`

Function and parameter names must begin with a lower-case letter

(Type names begin with an upper-case letter)

An *expression* can be

- a *literal* like `0` or `"xyz"`,

An *expression* can be

- a *literal* like `0` or `"xyz"`,

- or an *identifier* like `True` or `x`,

An *expression* can be

- a *literal* like `0` or `"xyz"`,
- or an *identifier* like `True` or `x`,
- or a *function application* $f\ e_1\ \ldots\ e_n$ where f is a function and $e_1\ \ldots\ e_n$ are expressions,
- or a parenthesized expression (e)

An *expression* can be

- a *literal* like `0` or `"xyz"`,
- or an *identifier* like `True` or `x`,
- or a *function application* $f\ e_1\ \ldots\ e_n$ where f is a function and $e_1\ \ldots\ e_n$ are expressions,
- or a parenthesized expression (e)

Additional syntactic sugar:

- `if then else`
- infix
- `where`
- ...

Local definitions: `where`

A defining equation can be followed by one or more local definitions.

Local definitions: `where`

A defining equation can be followed by one or more local definitions.

```
pow4 x  =  x2 * x2 where x2 = x * x
```

Local definitions: `where`

A defining equation can be followed by one or more local definitions.

```
pow4 x  =  x2 * x2 where x2 = x * x

pow4 x  =  sq (sq x) where sq x = x * x
```

Local definitions: `where`

A defining equation can be followed by one or more local definitions.

```
pow4 x  =  x2 * x2 where x2 = x * x

pow4 x  =  sq (sq x) where sq x = x * x

pow8 x  =  sq (sq x2)
  where x2 = x * x
        sq y = y * y
```

Local definitions: `let`

Local definitions: `let`

Local definitions: `let`

`let` x `=` e_1 `in` e_2

defines x locally in e_2

Local definitions: `let`

`let` x `=` e_1 `in` e_2

defines x locally in e_2

Example:

```
let x = 2+3 in x^2 + 2*x
= 35
```

Local definitions: `let`

`let` x `=` e_1 `in` e_2

defines x locally in e_2

Example:

```
let x = 2+3 in x^2 + 2*x
= 35
```

Like e_2 `where` x `=` e_1

TTT Fri 09:47

Example: `concat`

```
concat xss  =  [x | xs <- xss, x <- xs]

concat [[1,2], [4,5,6]]
= [x | xs <- [[1,2], [4,5,6]], x <- xs]
= [x | x <- [1,2]] ++ [x | x <- [4,5,6]]
= [1,2] ++ [4,5,6]
= [1,2,4,5,6]
```

What is the type of `concat`?

`[[a]] -> [a]`