ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Extended Example: Discrete Event Simulation

Principles of Reactive Programming

Martin Odersky

Extended Example: Discrete Event Simulation

Here's an example that shows how assignments and higher-order functions can be combined in interesting ways.

We will construct a digital circuit simulator.

The simulator is based on a general framework for discrete event simulation.

Digital Circuits

Let's start with a small description language for digital circuits.

A digital circuit is composed of *wires* and of functional components.

Wires transport signals that are transformed by components.

We represent signals using booleans `true` and `false`.

The base components (gates) are:

- The *Inverter*, whose output is the inverse of its input.
- The *AND Gate*, whose output is the conjunction of its inputs.
- The *OR Gate*, whose output is the disjunction of its inputs.

Other components can be constructed by combining these base components.

The components have a reaction time (or *delay*), i.e. their outputs don't change immediately after a change to their inputs.

Digital Circuit Diagrams

A Language for Digital Circuits

We describe the elements of a digital circuit using the following Scala classes and functions.

To start with, the class `Wire` models wires.

Wires can be constructed as follows:

```
val a = new Wire; val b = new Wire; val c = new Wire
```

or, equivalently:

```
val a, b, c = new Wire
```

Gates

Then, there are the following functions. Each has a side effect that creates a gate.

```
def inverter(input: Wire, output: Wire): Unit
def andGate(a1: Wire, a2: Wire, output: Wire): Unit
def orGate(o1: Wire, o2: Wire, output: Wire): Unit
```

Constructing Components

More complex components can be constructed from these.

For example, a half-adder can be defined as follows:

```
def halfAdder(a: Wire, b: Wire, s: Wire, c: Wire): Unit = {
  val d = new Wire
  val e = new Wire
  orGate(a, b, d)
  andGate(a, b, c)
  inverter(c, e)
  andGate(d, e, s)
}
```

More Components

This half-adder can in turn be used to define a full adder:

```scala
def fullAdder(a: Wire, b: Wire, cin: Wire, sum: Wire, cout: Wire): Unit = {
  val s = new Wire
  val c1 = new Wire
  val c2 = new Wire
  halfAdder(b, cin, s, c1)
  halfAdder(a, s, sum, c2)
  orGate(c1, c2, cout)
}
```

Exercise

What logical function does this program describe?

```scala
 def f(a: Wire, b: Wire, c: Wire): Unit = {
   val d, e, f, g = new Wire
   inverter(a, d)
   inverter(b, e)
   andGate(a, e, f)
   andGate(b, d, g)
   orGate(f, g, c)
}
```

O	a & ~b	O	a & ~(b & a)	O	b & ~a
O	a == b	O	a != b	O	a * b