Using Logic for Programming Reactive Systems

Wolfgang Jeltsch

TTÜ Küberneetika Instituut

TTÜ Küberneetika Institudi sügisseminar

November 13, 2011

Propositional logic

- basic operators:

  $\alpha \wedge \beta$ α and β

  $\alpha \vee \beta$ α or β

  $\alpha \to \beta$ if α then β

  $\bot$ false

  $\top$ true

- derived operators:

  $\alpha \leftrightarrow \beta$ α if and only if β

  $$\alpha \leftrightarrow \beta := (\alpha \to \beta) \wedge (\beta \to \alpha)$$

  $\neg\alpha$ not α

  $$\neg\alpha := \alpha \to \bot$$

Simple types

- operators:

$\alpha \times \beta$	type of pairs that consist of an α-value and a β-value
$\alpha + \beta$	type of variants that are either an α-value or a β-value
$\alpha \to \beta$	type of functions from α to β
0	empty type
1	singleton type

Curry–Howard correspondence

- correspondence between logic and type system:

$$\begin{array}{ccc} \text{proposition} & \longleftrightarrow & \text{type} \\ \uparrow & & \uparrow \\ \text{proof} & \longleftrightarrow & \text{expression} \end{array}$$

- well-known correspondences:
  - propositional logic $\longleftrightarrow$ simple types:

$$\begin{aligned} \langle \alpha \wedge \beta \rangle &= \langle \alpha \rangle \times \langle \beta \rangle \\ \langle \alpha \vee \beta \rangle &= \langle \alpha \rangle + \langle \beta \rangle \\ \langle \alpha \to \beta \rangle &= \langle \alpha \rangle \to \langle \beta \rangle \\ \langle \bot \rangle &= 0 \\ \langle \top \rangle &= 1 \end{aligned}$$

  - predicate logic $\longleftrightarrow$ dependent types:
    - almost any specification expressible as a type
    - specifications checked by the compiler

Temporal logic

- lies between propositional and predicate logic:
  - not as expressible as predicate logic
  - easier to handle than predicate logic
- trueness of a proposition depends on the time
- all operators from propositional logic plus a few new ones:

  $\Box\alpha$ α will always hold

  $\Diamond\alpha$ α will eventually hold

  $\alpha \triangleright \beta$ α will hold for some time, and then β will hold

  $\bigcirc\alpha$ α will hold at the next point in time
- discrete vs. continuous time:
  - $\bigcirc$ requires a discrete time scale
  - by dropping $\bigcirc$, we can also handle continuous time

A Curry–Howard correspondence for temporal logic

- type membership depends on the time
- all simple type operators plus a few new ones:

■ ◆ ► ●

- a type member denotes a proof of the corresponding logical formula:

  $\blacksquare\alpha$ a family of α-values, one for each future time

  $\blacklozenge\alpha$ a future time togeter with an α-value

  $\alpha \blacktriangleright \beta$ a tuple of the following:
  - a future time (the “stop time”)
  - a family of α-values, one for each future time before the stop time
  - a β-value, belonging to the stop time

  $\bullet\alpha$ an α-value, belonging to the next time

Does this look familiar?

- ■ and ◆ are essentially type operators from FRP (Functional Reactive Programming)
- FRP:

  a declarative way of programming reactive systems
- two core concepts:

  behaviors time-varying values

  events values occurring at a certain point in time
- examples:

  behavior position of the mouse

  event a key press
- executable system descriptions built by composing behaviors and events
- ■ and ◆ construct behavior and event types, respectively

What about the other operators?

- ▶-values are finite behaviors with a terminating event
- examples:
  - an audio signal that terminates at some point:

$$(\mathbb{R} \times \mathbb{R}) \blacktriangleright 1$$

  - an audio signal that switches between stereo and mono:

$$(\mathbb{R} \times \mathbb{R}) \blacktriangleright \mathbb{R} \blacktriangleright (\mathbb{R} \times \mathbb{R}) \blacktriangleright \mathbb{R} \blacktriangleright \ldots$$

    (assuming ▶ is right-associative)
  - an audio signal that is sometimes interrupted:

$$(\mathbb{R} \times \mathbb{R}) \blacktriangleright 1 \blacktriangleright (\mathbb{R} \times \mathbb{R}) \blacktriangleright 1 \blacktriangleright \ldots$$

    etc.
- ●-values for “Functional Reactive Dataflow Programming”
- dataflow programming:

  working with streams of values associated with discrete times

Conclusions and outlook

- Curry–Howard correspondence as a one-to-one mapping between concepts from logic and concepts from programming
- programming can benefit from logic and vice versa
- discovery of the Temporal Curry–Howard Correspondence leads to improvements of FRP
- examples shown in this talk:
  - ▶-types for specifying advanced temporal behavior
  - ●-types for bringing the glory of FRP to dataflow programming
- further examples:
  - a principled way of ensuring start time consistency for avoiding problems with semantics and performance
  - a better structured interface to FRP
  - new techniques for implementing FRP