Pure and Faust: Functional Programming for Media Applications

Albert Gräf
Department of Music Informatics
Johannes Gutenberg University Mainz

JG|U
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Overview

- Term rewriting as a model of computation
- A brief tour of Pure
- Building a synthesizer with Pure and Faust
- Interfacing to Pd
- Demo of pd-faust
- Conclusion

Models of computation

- Recursive functions $\Rightarrow$ theory of computation
- String rewriting (Semi-Thue) systems $\Rightarrow$ Turing machines, grammars
- Term rewriting $\Rightarrow$ universal algebra, computer algebra, algebraic specification
- Lambda calculus $\Rightarrow$ functional programming
- Random access machine $\Rightarrow$ von Neumann architecture, imperative programming

Advantages of TR

- Purely functional (no side effects)
- Allows tree-like data structures
- More expressive than lambda calculus
- No distinction between defined functions and data constructors
- Symbolic evaluation (computer algebra)
- Constructor equations

```
> x:y:xs = y:x:xs if x>y; x:y:xs = x:xs if x==y;
> [13,7,9,7,1]+[1,9,7,5];
[1,5,7,9,13]
```

Term rewriting

- Signature Σ (alphabet with arities) of function and variable symbols
- Terms $f\, t_1 \ldots t_n\ (f \in \Sigma_n) \Rightarrow$ term algebra $T(\Sigma)$
- Alternate representations: labelled trees, term DAGs (directed acyclic graphs) $\Rightarrow$ graph rewriting
- Rewriting rules are of the form $p \rightarrow q,\ p,q \in T(\Sigma)$.

Term reductions

- R = finite set of term rewriting rules
- variable substitutions: σ = $[x_1 \to t_1, x_2 \to t_2, \ldots]$
- reduction step: $t[\sigma(p)] \to_R t[\sigma(q)]$, $p \to q \in R$
- $\sigma(p)$ is called the **redex**, $\sigma(q)$ the **reduct**, σ the **matching substitution**
- $u \to^*_R v$ if u reduces to v in zero or more steps (reflexive/transitive closure)
- if v is irreducible, it is a **normal form** of u

Reduction strategy

- Order in which redices are used and rules are picked matters if R isn't both confluent and terminating
- Optimal (or at least terminating) reduction strategies are known for some systems, but are undecidable in general
- Practical solution: rule order + redex selection strategy
- Leftmost-innermost (eager, “call by value”)
- Leftmost-outermost (lazy, “call by need”)

Term rewriting as a PL

- Mike O'Donnell 1985: Equational Logic as a Programming Language
- Computer algebra, algebraic programming (OBJ, OPAL) (for special purposes)
- 1991: Q
- 1994: Mozart/Oz (CTMCP book)
- 2000: Aardappel (Oortmerssen)
- 2008: Pure

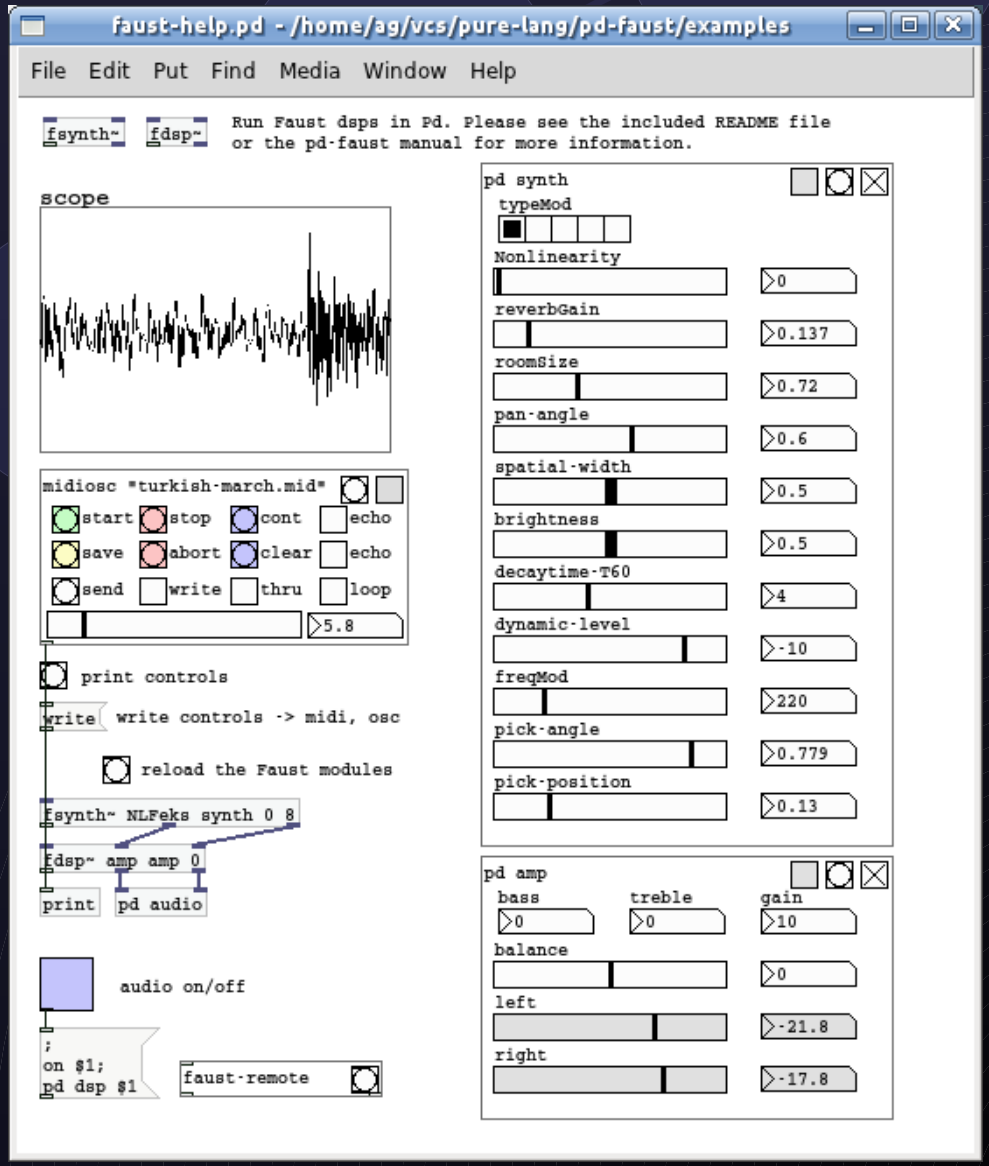
Pure as a term rewriting language

- Purely functional core + ability to call any C function (and thereby have side effects)
- Conditional and ordered rewriting
- Leftmost-innermost (eager) evaluation, lazy evaluation via “thunks”
- Lambdas, local functions and variables (block constructs with lexical scoping)
- FP-style currying and partial applications
- Types as predicates, interface types

Additional features

- Interactive interpreter-like environment
- JIT (just in time) compilation to native code using the LLVM toolkit (Lattner et al)
- Batch compilation
- Easy interface to C/C++, Fortran, Faust, Octave, ...
- Programming modes for emacs, vi et al
- Compiled functional scripting language

Demo



JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Conclusion

- Pure: a general-purpose FPL based on term rewriting
- High-level programming style
- Easy interface to C, C++, Fortran and Faust
- Interfaces to MIDI, audio, OSC, Pd, Octave, Gnumeric ...
- Use as a glue language, as a compiled functional scripting language, for doing control stuff ...
- Try for yourself: “Planet Pure+Faust” Ubuntu 11.04 LiveCD with all things Pure+Faust already installed http://download.linuxaudio.org/pure+faust/