N+1 CHALLENGES FOR SLE

Friedrich Steimann@OOPSLE 2020
Fernuniversität in Hagen
Germany

outline

- give up trees
- tend to your meta-language
- the metalevel is not the world of static
- diversify
- escape the empiricism trap
- …
- +1

GIVE UP TREES

or: embrace the modelling community

give up trees

- not everything is a tree
- just like not everything is a function
- for everything is in fact a relation
- just like everything is a graph

- in modelling, a tree is just a degenerate graph
  - *undercut only by lists*

give up trees

trees vs. graphs

PL vs. SE (Albert Zündorf)

mathematical rigour vs. empiricism

hard science vs. soft science

give up trees

- but trees are great
- inductive definitions
- inductive proofs

J. Functional Programming 11 (5): 467–492, September 2001. Printed in the United Kingdom 467
© 2001 Cambridge University Press

Inductive graphs and functional graph algorithms

MARTIN ERWIG

Department of Computer Science, Oregon State University, Corvallis, Oregon 97331, USA
(e-mail: erwig@cs.orst.edu)

Abstract

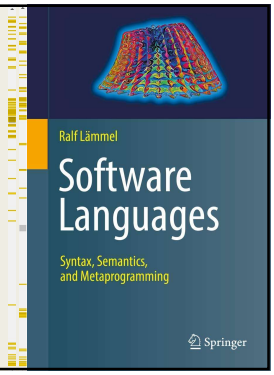
We propose a new style of writing graph algorithms in functional languages which is based on an alternative view of graphs as inductively defined data types. We show how this graph model can be implemented efficiently, and then we demonstrate how graph algorithms can be succinctly given by recursive function definitions based on the inductive graph view. We also regard this as a contribution to the teaching of algorithms and data structures in functional languages since our style of presenting functional-style graph algorithms instead of the imperative algorithms that are dominant today.

1 Introduction

How should I implement a graph algorithm in a functional programming language? This seemingly simple question has attracted attention for quite a long time, and there are many different proposals for how to do so. Of course, it is not really difficult to *somehow* realize graph algorithms in functional languages. The real challenge is to obtain *clear* and *efficient* programs, that is, functional programs that do not lose their elegance and simplicity and that have the same asymptotic complexity as imperative ones.

The main difficulties that arise when dealing, for example, with depth-first search

so, why not use graphs



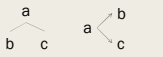
conjecture:

the linear nature of our language(s) makes us prefer trees over graphs

text bla bla *a*(*b*, *c*) bla bla text

text bla bla {(*a*, *b*), (*a*, *c*)} bla bla text

needs building up a mental environment (dictionary)



a(*b*, *a*) {(*a*, *b*), (*a*, *a*)}

a(*b*, "*a*")

The Graph Paradox:
In working with graphs, we resort to trees.

(i.e., we express the general in terms of the special, while it should be the other way round)

caused by language

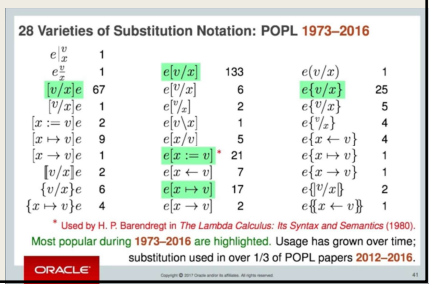
give up trees

- trees go with functions
- like graphs go with relations
- has SLE a preference for functional metalanguages?
- adoption of functional PL constructs in language definitions and proofs
  - *Maybe and Either monads in Isabelle*
  - *List monads, too?*
  - *not needed in relational languages?*
  - *are functions a source of accidental complexity?*
- if the linearity of metalanguages suggests trees, is progress slowed by the linearity of metalanguages?

TEND TO YOUR META-LANGUAGE

tend to your metalanguage

Guy Steele:
"It's Time for a New Old Language"
(2017 ACM PPoPP Keynote)



tend to your metalanguage

e^* $e_1 \dots e_n$ $\overline{e}$

- what are these?
  - *"unenclosed sequence", "unpackaged collection"*
- that we can still read this texts mean high redundancy
  - *can squeeze even more on a page*
- common metalanguage?
- LaTeX!
  - *but that's a meta-metalanguage*
- how about a standard metalanguage?

USLES

tend to your metalanguage

- how about choosing one with precise syntax and semantics?
- how about an executable one?
- theorems and proofs about properties of the object language …
- … become program verification (verification of the interpreter)

{P} S {R}

- how about a programming language as metalanguage?
  - *Haskell?*
  - *Coq?*

how about PROLOG?

$$\frac{(S, e_1) \Rightarrow (S'', v_1) \quad (S'', e_2) \Rightarrow (S', v_2) \quad v = v_1 + v_2}{(S, e_1 + e_2) \Rightarrow (S', v)}$$

```
(S, E1 + E2) => (S1, V) :-
    (S, E1) => (S2, V1),
    (S2, E2) => (S1, V2),
    V is V1 + V2.

:- ([x=1, y=2], x+y) => (S, V).
yes: S = [x=1, y=2], V = 3

:- ([x=1, y=2],x*y) => (S, V).
no
:- ([x=1],x+y) => (S, V).
no
```

$$\frac{(S, e) \Rightarrow (S'', v) \quad \mathrm{def}(f(x)) = e' \quad (S'', \left[\frac{x}{v}\right] e') \Rightarrow (S', v')}{(S, f(e)) \Rightarrow (S', v')}$$

```
(S, f(F, E)) => (S1, V1) :-
    (S, E) => (S2, V),
    def(F, V, E1),
    (S2, E1) => (S1, V1).

:- ([z=3], fak(z)) => (S, V).
yes: S = [z=3], V = 6

:- ([z=-1], fak(z)) => (S, V).
…
```

is PROLOG relational?

?- f(X) = Y, g(Y) = X.

with occurs-check turned on?

THE METALEVEL IS NOT THE WORLD OF STATIC

or: make editing a discipline

the metalevel is not the world of static

- static semantics (strong typing) are standard on the metalevel
- what about dynamic semantics?
- dynamic semantics for
  - *transformation (sure, the compiler)*
  - *editing*
  - *synthesis*
  - *(incl. refactoring)*
- with various soundness guarantees for the object level
  - *change a program, guarantee that it still compiles*
  - *change a language and its programs with it, with strong guarantees*

DIVERSIFY

or: software language tools are not the only programs of interest

SLE
=
the metacircular science of metacircularity

we live on our own dog food

diversify

conjecture:

For a substantial number of students,
a compiler is the only program they have ever written or even seen.

They view everything as a function/tree.

diversify

- ok for research in SLE
- sufficient for having impact?
- 80% of software is embedded
  - *security and realtime as important as safety*
  - *persistence, distribution, and roll-back*
- compared with medicine (engineering), SLE has no impact over what is happening in the wild
  - *medical societies control everything related to diagnosis and treatment*
  - *we control nothing*
  - *even though humankind is subjected to programmers' doing*

diversify

conjecture:

To get impact, we need to embrace all software.

ESCAPE THE EMPIRICISM TRAP

or: let's formalize utility

escape the empiricism trap

- from SE, student experiments, questionaires
- instead find and establish good analytical models of judging the utility of SLE tools
- not number of keystrokes, but perhaps number of decisions to be made
- (game theory)
- validate the model once, then work with it forevermore
- (like the mouse model in pharmaceutics)
- good models are hard to find (corona)
- field of study is language
- not a specific language, not even all exisiting languages, but all languages that can be conceived of
- infinite study space
- make the best of this freedom
- stop doing things in a way that "because the proofs are easier" that's a road to insignificance

escape the empiricism trap

- medicine is evidence-based
  - *doctors want proofs*
- our formally proving is great
- but restricts progress to the formally provable
- drives progress to the formally provable
  - *I know how to prove it, so I propose it*
- how do I prove that *x* is good or bad?

escape the empiricism trap

- empirical experiments
  - *hard to do*
- all our empirical experiments are psychological experiments
- confounding factors hard to control
- its mathematics are unknown (for many of us)
- generalization is hard to show

escape the empiricism trap

- medical (pharmaceutical) research works with models
  - *mouse model*
- each new treatment requires a new study on humans
- each new treatment requires a new study on mice
- one study showing that studies on mice allow predictions for humans
- we can't get mice to code

escape the empiricism trap

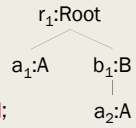
- programming with tools is a game
  - *with players* programmer *and* tools
  - *each taking turns*
    - tools provide clues
    - programmer decides
- programming = working off one giant decision graph (or tree?)
- find mathematical models
- show how tools reduce the size of the graph
- one final experiment showing that model is valid

HOW TO TYPE THAT DAMN PARENT ATTRIBUTE?

or: if you can't type this, what can you do?

how to type that damn parent attribute?

```
node type Root begin a : A; b : B end;
node type A begin parent() : Root end;
node type B begin a : A; parent() : Root end;
```



a_1.parent().parent()	static error
a_2.parent().parent()	static error