Domain Specific Languages
5: Are DSLs domain specific?

Greg Michaelson

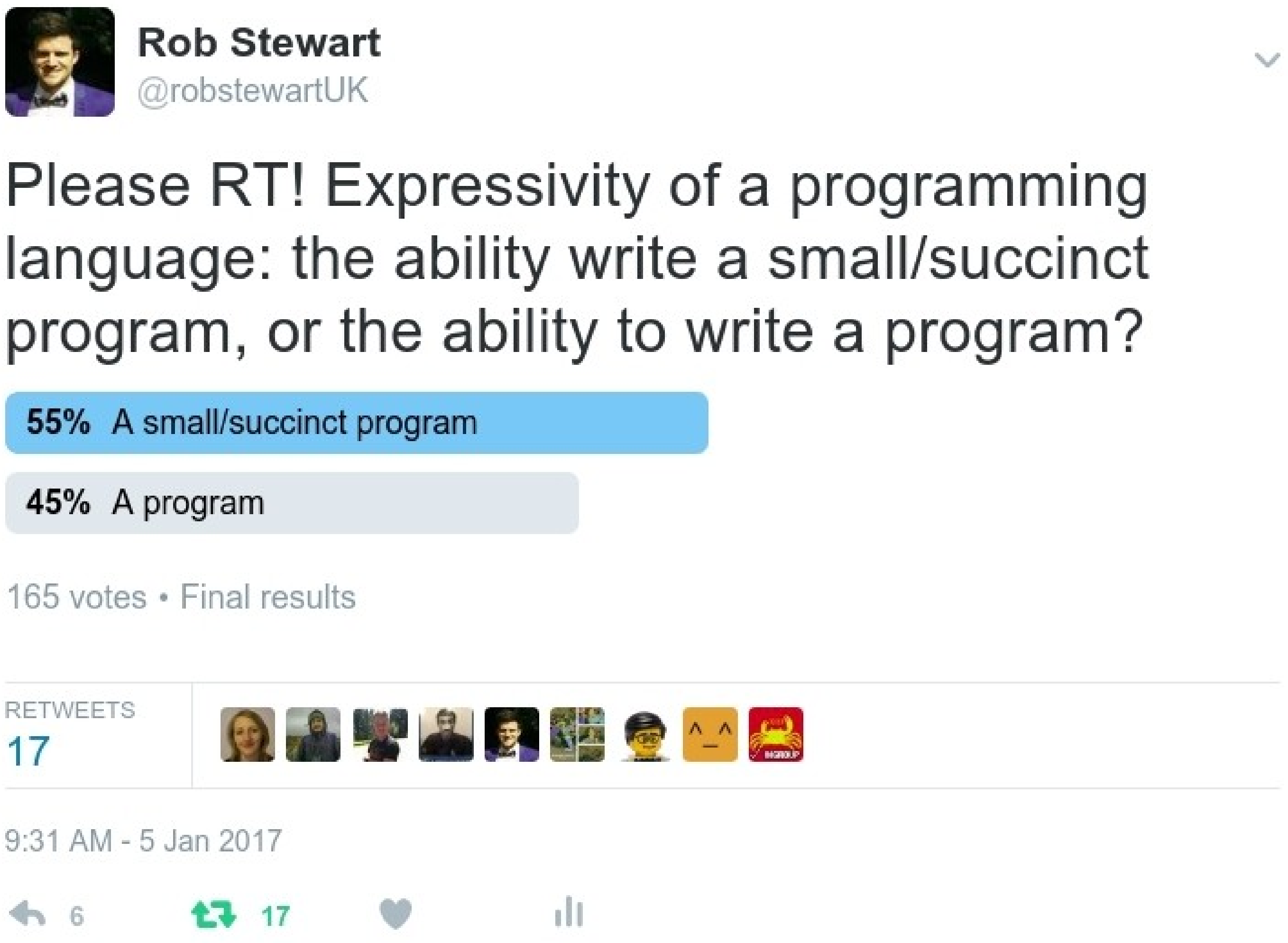
School of Mathematical & Computer Sciences

Heriot Watt University

DSLs and expressiveness

- DSL’s alleged to have greater *expressiveness* than general purpose languages
- claimed DSLs can express:
  - same things as other languages, but more *succinctly*
  - things other languages can’t
- notion of expressiveness is *comparative*

What is expressivity?



Rob Stewart
@robstewartUK

Please RT! Expressivity of a programming language: the ability write a small/succinct program, or the ability to write a program?

55% A small/succinct program

45% A program

165 votes • Final results

RETWEETS
17

9:31 AM - 5 Jan 2017

Formalising expressiveness

- M. Felleisen, On the Expressive Power of Programming Languages, *Science of Computer Programming*, pp134-151, 1990
- suppose language X has constructors which are not in language Y
- language X is a *conservative extension* of language Y if:
  - instances of X can be translated into instances of Y without changing semantics of Y

Formalising expressiveness

- e.g. GpH adds `par` & `seq` constructs to Haskell
  - conservative extension
  - simple elimination of `seq` & `par` preserves meanings of programs
- *weak expressibility*

Formalising expressiveness

- if constructors in X cannot be eliminated in translation to Y then:
  - X has semantic properties Y lacks
  - X *extends* Y
- e.g. pure functional Scheme based on LISP
- cannot express state changing assignment `set!`
- LISP extends Scheme

Embedded DSL (EDSL)

- DSL embedded in extant host language

1. write functions in host language

- call directly
- can use arbitrary host language constructs
- not a distinct language
- *shallow embedding*

Embedded DSL

2. add abstract syntax

- interpret ASTs
- call interpreter components from arbitrary host language constructs
- not a distinct language unless only use ASTs to program
- *deep embedding*
- conservative extension

Embedded DSL

3. design concrete syntax

- add concrete syntax -> AST compiler
- call interpreter component with parsed strings
- can embed interpreter component calls in arbitrary host language constructs
- not a distinct language unless only use concrete syntax strings
- *deep embedding*
- conservative extension

Embedded DSL

4. extend host language syntax

- conservative extension

5. extend host language semantics

- extension

Turing completeness

- Hilbert’s program
  - can number theoretic predicate calculus establish its own completeness & consistency?
    - no – Gödel - 1932
  - can theorem-hood be established mechanically?
    - no - Turing & Church - 1936

Turing completeness

- need a system for formalising “mechanical”
- models of computation
  - Church
    - algorithm/effectively calculable
    - λ calculus
  - Turing
    - computable
    - Turing machines
- Turing proved these equivalent 1936/37

Turing completeness

- Church-Turing thesis
  - all models of computation are equivalent
  - demonstrate by constructing/proving translations both ways between known & new systems
- system that satisfies C-T Thesis is *Turing complete* (*TC*)
- e.g. von Neumann machines ㅍ digital computers
- e.g. programming languages

EDSL & host language

- EDSL inherits semantics of host language
- if host language is not TC then EDSL may be more expressive
  - i.e. EDSL == host + external library in 3rd language
  - can't be less or differently expressive
- if host language is TC then so is EDSL
  - not domain specific?

EDSL & host language

- if host language exposed to programmer…
- …then programmer can deploy arbitrary host language constructs
- not domain specific?

Implemented DSL

- expose parser/interpreter only as *stand alone* language processor
- can only use domain specific syntax
- have *implemented* DSL in host language

DSL bloat

- tend to want familiar general purpose programming language abstractions as well, so add:
  - arithmetic & logic
  - sequence/selection/iteration
  - sub programs
  - data structures
- as DSL grows, tends to become :
  - more and more like favourite language
  - less and less DS

Language & program

- language = syntax + semantics
- semantics: program * state -> state
- semantics transforms initial state to final state guided by structure of program instance

Language & program

- program: input * state -> output * state
- treat outputs as part of final state
- program: input * state -> state
- program changes initial state to final state depending on input structure
- define input structure with syntax
- program is semantics of inputs
- does every program define a DSL?

What is domain specificity?

- Light Bulb Language

program -> *switch*

switch -> SWITCH |ε

m [SWITCH] ON = OFF

m [SWITCH] OFF = ON

m [ε] ON = ON

m [ε] OFF = OFF

What is domain specificity?

- Linear Light Bulbs Language

program -> *row* | *row program*

row -> *switch* | *switch row*

$s_i \in$ *row*

$b_i \in \{\mathrm{ON},\mathrm{OFF}\}$

$m' \, [s_1,\ldots,s_N] \, \{b_1,\ldots,b_N\} = \{m \; s_1 \; b_1,\ldots,m \; s_N \; b_N\}$

What is domain specificity?

- Grid Light Bulbs Language

program -> grid

grid -> row | row grid

r_i ϵ *row*

br_i ϵ b*

m’’ [r_1,…,r_N] {br_1,…,br_N} = {m’ p br_1,…,m’ p br_N}

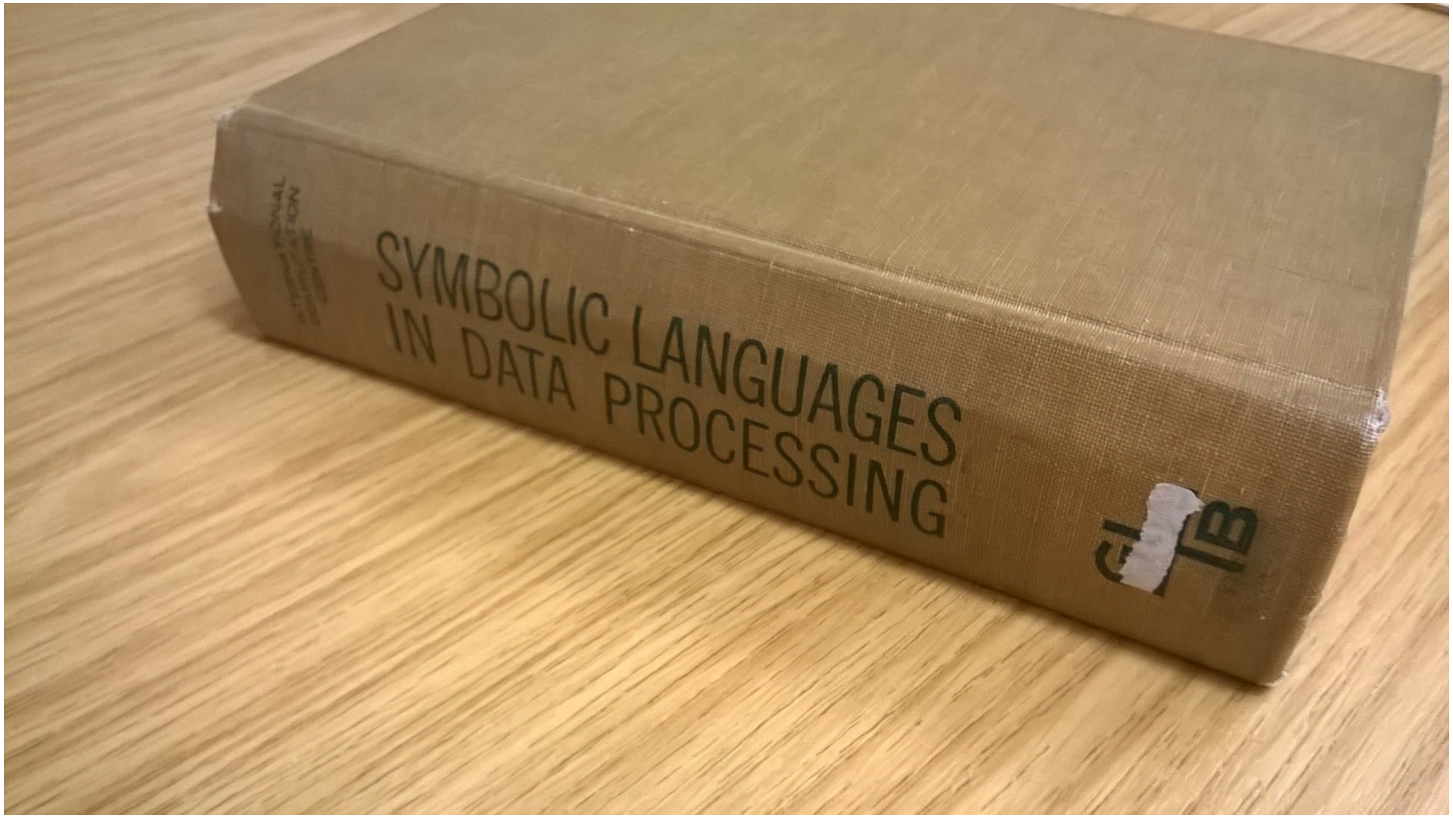
What is domain specificity?

- is GLBL domain specific…?
  - light bulbs?
  - B/W images?
  - anything representable as a Boolean?
- not very DS…

What is a domain?

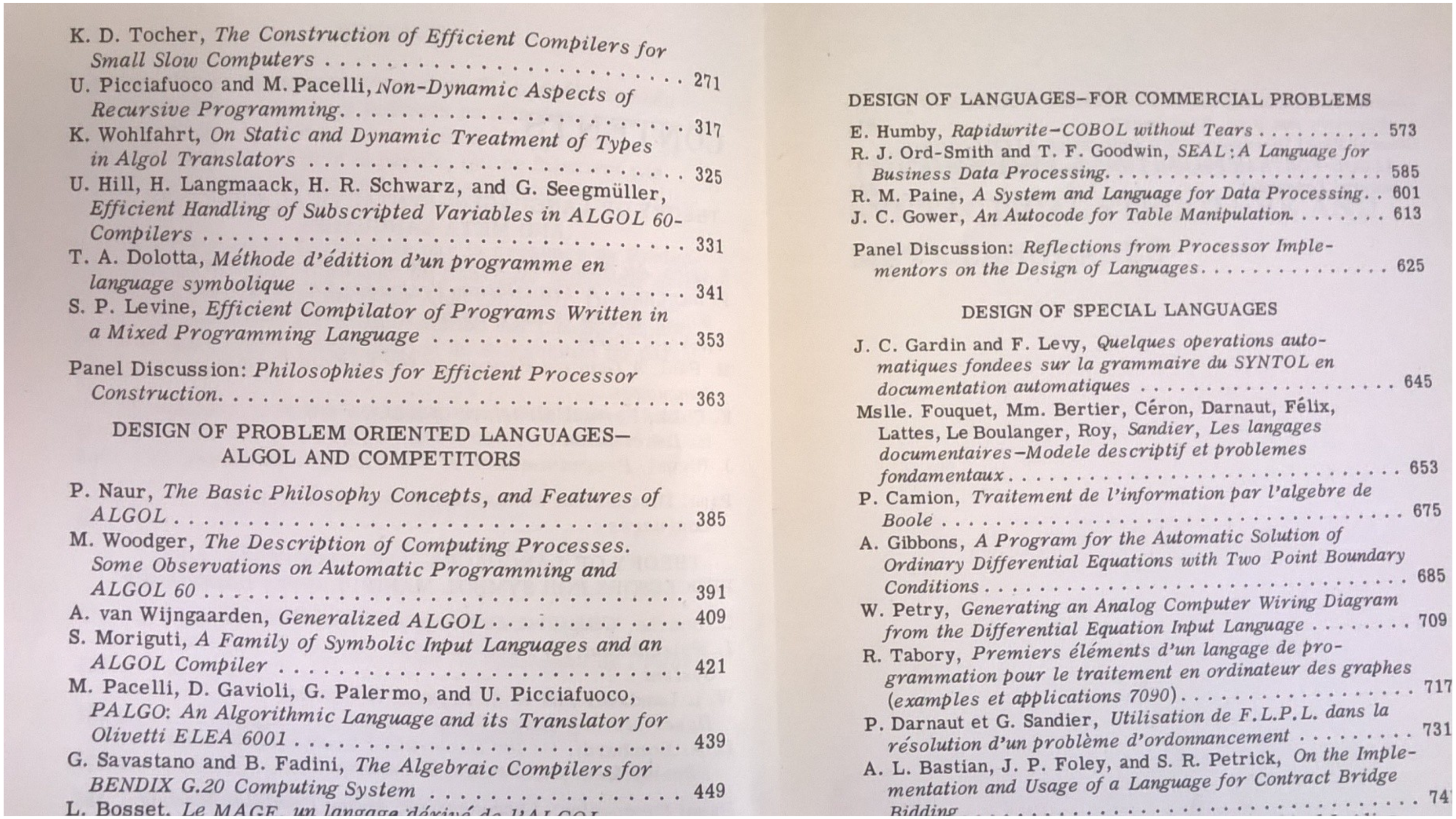
- computer pioneers thought machine code was for configuring hardware
- 60’s language designers thought languages were purpose specific:
  - FORTRAN - sums
  - COBOL - accounts
  - ALGOL - algorithms
  - LISP - symbols
  - BCPL - systems

What is a domain?



1962

What is a domain?



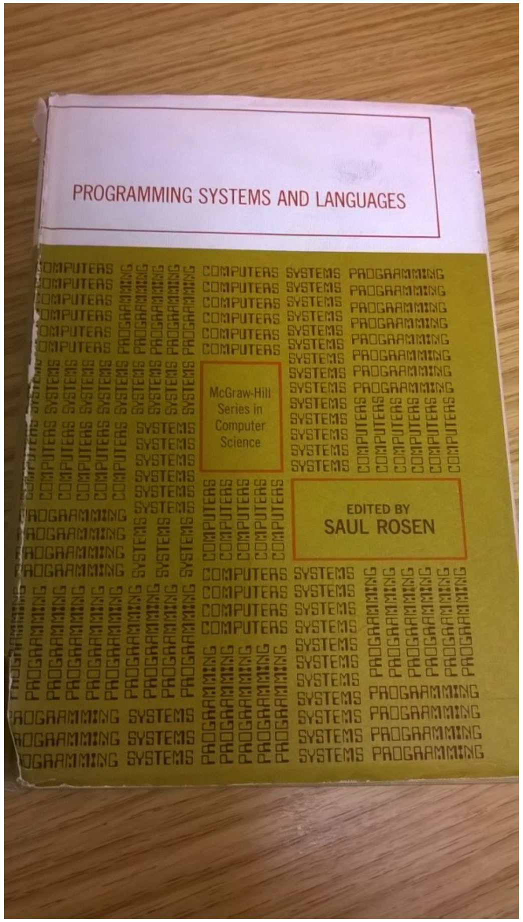
DESIGN OF PROBLEM ORIENTED LANGUAGES—ALGOL AND COMPETITORS

DESIGN OF LANGUAGES–FOR COMMERCIAL PROBLEMS

DESIGN OF SPECIAL LANGUAGES

1962

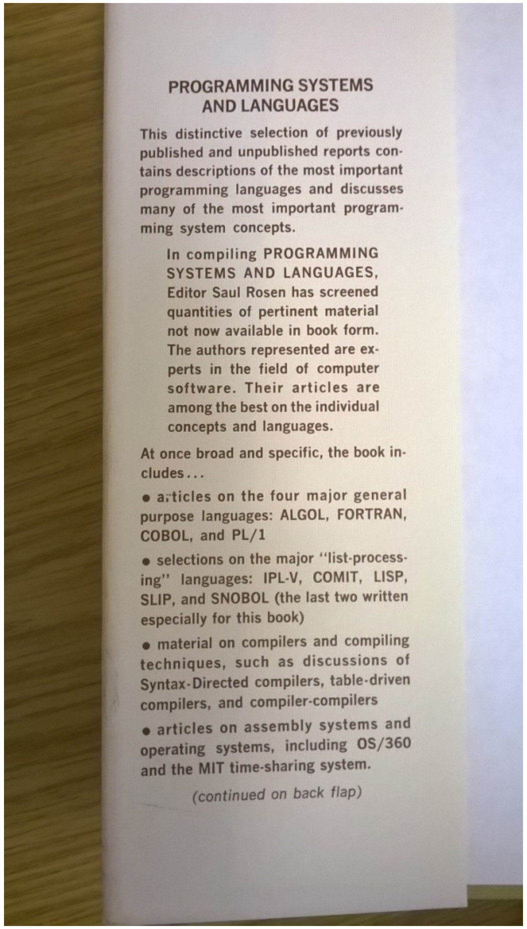
What is a domain?



PROGRAMMING SYSTEMS AND LANGUAGES

McGraw-Hill Series in Computer Science

EDITED BY SAUL ROSEN



PROGRAMMING SYSTEMS AND LANGUAGES

This distinctive selection of previously published and unpublished reports contains descriptions of the most important programming languages and discusses many of the most important programming system concepts.

In compiling PROGRAMMING SYSTEMS AND LANGUAGES, Editor Saul Rosen has screened quantities of pertinent material not now available in book form. The authors represented are experts in the field of computer software. Their articles are among the best on the individual concepts and languages.

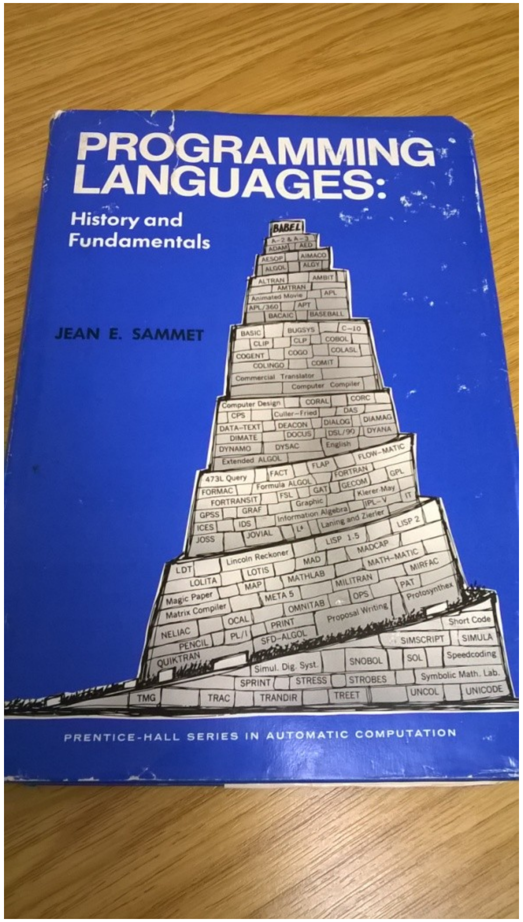
At once broad and specific, the book includes...

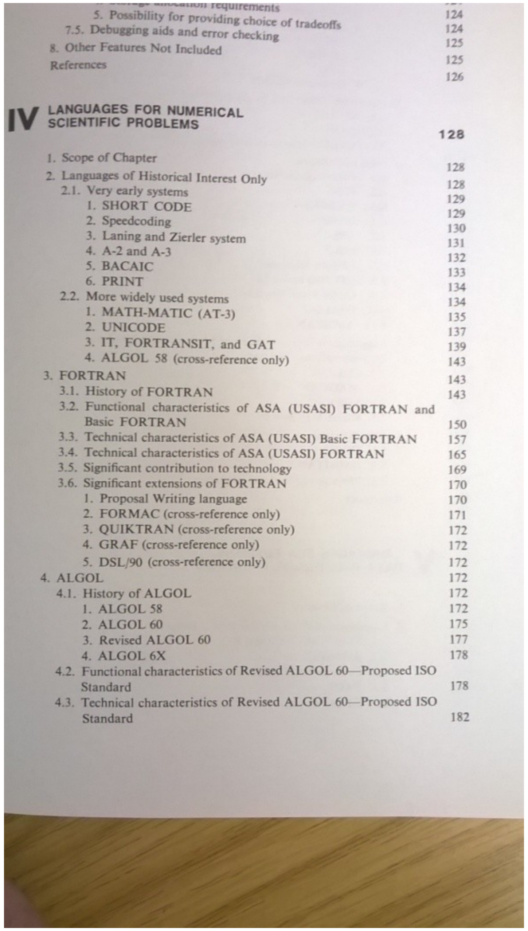
- articles on the four major general purpose languages: ALGOL, FORTRAN, COBOL, and PL/1
- selections on the major "list-processing" languages: IPL-V, COMIT, LISP, SLIP, and SNOBOL (the last two written especially for this book)
- material on compilers and compiling techniques, such as discussions of Syntax-Directed compilers, table-driven compilers, and compiler-compilers
- articles on assembly systems and operating systems, including OS/360 and the MIT time-sharing system.

(continued on back flap)

1964

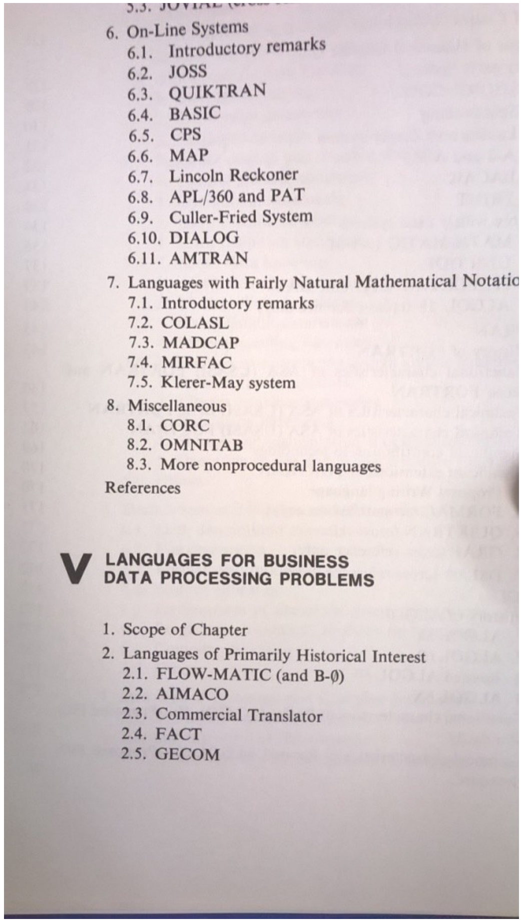
What is a domain?





1969

What is a domain?



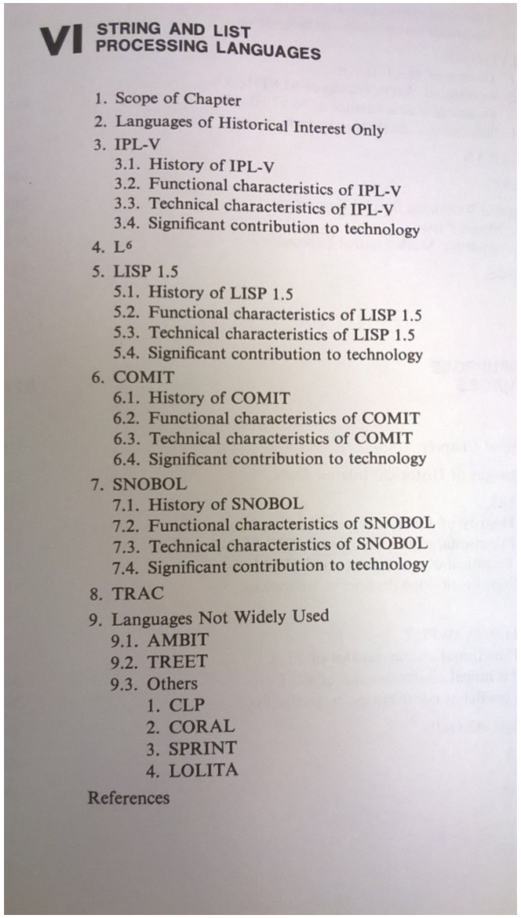
5.3. JOVIAL (cross

6. On-Line Systems
   - 6.1. Introductory remarks
   - 6.2. JOSS
   - 6.3. QUIKTRAN
   - 6.4. BASIC
   - 6.5. CPS
   - 6.6. MAP
   - 6.7. Lincoln Reckoner
   - 6.8. APL/360 and PAT
   - 6.9. Culler-Fried System
   - 6.10. DIALOG
   - 6.11. AMTRAN
7. Languages with Fairly Natural Mathematical Notatio
   - 7.1. Introductory remarks
   - 7.2. COLASL
   - 7.3. MADCAP
   - 7.4. MIRFAC
   - 7.5. Klerer-May system
8. Miscellaneous
   - 8.1. CORC
   - 8.2. OMNITAB
   - 8.3. More nonprocedural languages

References

V LANGUAGES FOR BUSINESS DATA PROCESSING PROBLEMS

1. Scope of Chapter
2. Languages of Primarily Historical Interest
   - 2.1. FLOW-MATIC (and B-0)
   - 2.2. AIMACO
   - 2.3. Commercial Translator
   - 2.4. FACT
   - 2.5. GECOM



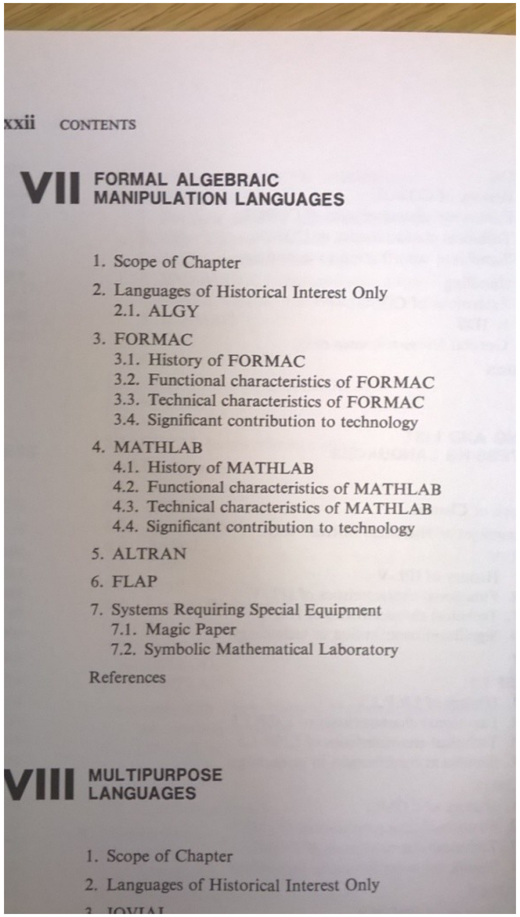
VI STRING AND LIST PROCESSING LANGUAGES

1. Scope of Chapter
2. Languages of Historical Interest Only
3. IPL-V
   - 3.1. History of IPL-V
   - 3.2. Functional characteristics of IPL-V
   - 3.3. Technical characteristics of IPL-V
   - 3.4. Significant contribution to technology
4. L^6
5. LISP 1.5
   - 5.1. History of LISP 1.5
   - 5.2. Functional characteristics of LISP 1.5
   - 5.3. Technical characteristics of LISP 1.5
   - 5.4. Significant contribution to technology
6. COMIT
   - 6.1. History of COMIT
   - 6.2. Functional characteristics of COMIT
   - 6.3. Technical characteristics of COMIT
   - 6.4. Significant contribution to technology
7. SNOBOL
   - 7.1. History of SNOBOL
   - 7.2. Functional characteristics of SNOBOL
   - 7.3. Technical characteristics of SNOBOL
   - 7.4. Significant contribution to technology
8. TRAC
9. Languages Not Widely Used
   - 9.1. AMBIT
   - 9.2. TREET
   - 9.3. Others
     1. CLP
     2. CORAL
     3. SPRINT
     4. LOLITA

References

1969

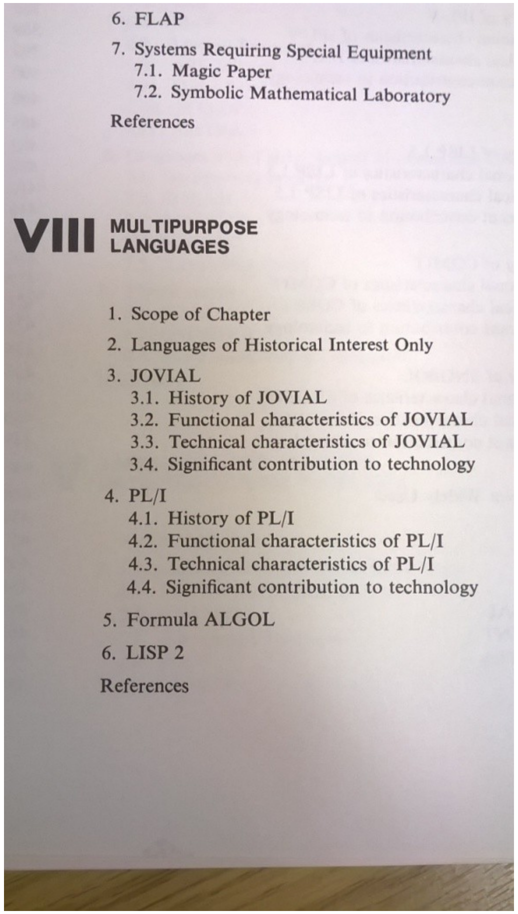
What is a domain?



xxii CONTENTS

VII FORMAL ALGEBRAIC MANIPULATION LANGUAGES

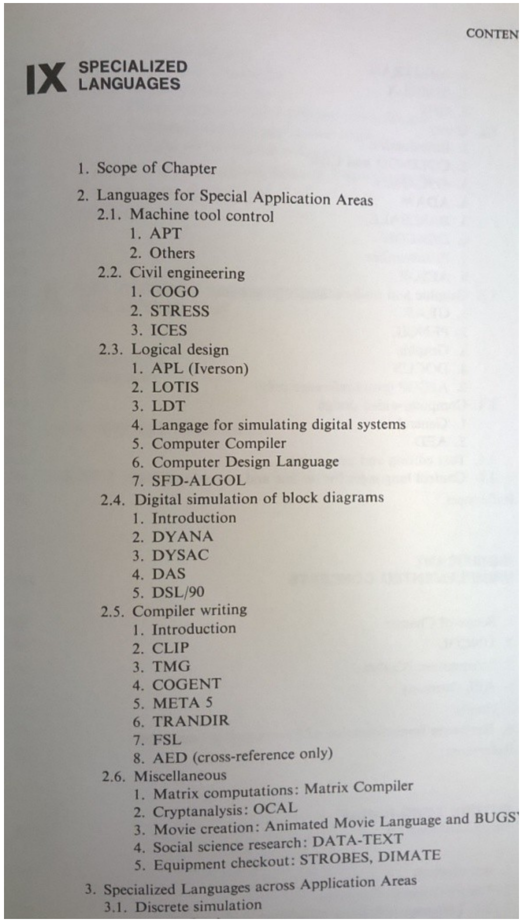
VIII MULTIPURPOSE LANGUAGES



VIII MULTIPURPOSE LANGUAGES

1969

What is a domain?



CONTEN

IX SPECIALIZED LANGUAGES

1. Scope of Chapter
2. Languages for Special Application Areas
   - 2.1. Machine tool control
     1. APT
     2. Others
   - 2.2. Civil engineering
     1. COGO
     2. STRESS
     3. ICES
   - 2.3. Logical design
     1. APL (Iverson)
     2. LOTIS
     3. LDT
     4. Langage for simulating digital systems
     5. Computer Compiler
     6. Computer Design Language
     7. SFD-ALGOL
   - 2.4. Digital simulation of block diagrams
     1. Introduction
     2. DYANA
     3. DYSAC
     4. DAS
     5. DSL/90
   - 2.5. Compiler writing
     1. Introduction
     2. CLIP
     3. TMG
     4. COGENT
     5. META 5
     6. TRANDIR
     7. FSL
     8. AED (cross-reference only)
   - 2.6. Miscellaneous
     1. Matrix computations: Matrix Compiler
     2. Cryptanalysis: OCAL
     3. Movie creation: Animated Movie Language and BUGS
     4. Social science research: DATA-TEXT
     5. Equipment checkout: STROBES, DIMATE
3. Specialized Languages across Application Areas
   - 3.1. Discrete simulation

1969

What is domain specificity?

- all TC languages capture common notion of computation
- TC languages with different semantics are mutually extending
- do all TC languages have the same expressiveness?

What is domain specificity?

- designers of new languages think they’ve enabled something other languages can’t do
- new TC language ≡

  old TC language + syntax + library
- is every language really an embeded DSL with a TC host?

What is domain specificity?

- Felleisen suggests comparisons in a common *language universe*
- some TC languages can express some algorithms more succinctly than other TC languages
- which language universe?
- do language universes have language biases?

DSL is about pragmatics

- Felleisen:
  - “…what advantages there are to programming in the more expressive language when equivalent programs in the simpler language already exist.”
  - “…programs in less expressive languages exhibit repeated occurrences of programming patterns and this pattern oriented style is detrimental to the programming process.”
  - “**Conciseness conjecture.** *Programs in more expressive languages that use the additional facilities in a sensible manner contain fewer programming patterns than equivalent programs in less expressive languages.*”

DSL is about pragmatics

- DSL abstractions & constructs make it easier to *express* particular things
- what may be complex in an arbitrary TC language may become simpler in a DSL
- domain may frame choice of DSL abstractions & constructions
- abstractions and constructions from one domain may be appropriate for other domains