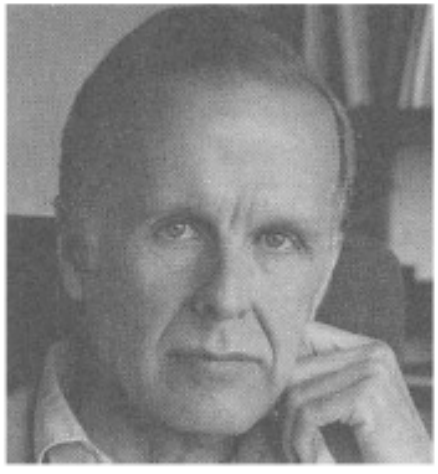
Une introduction au langage Haskell

Jonathan Laurent

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



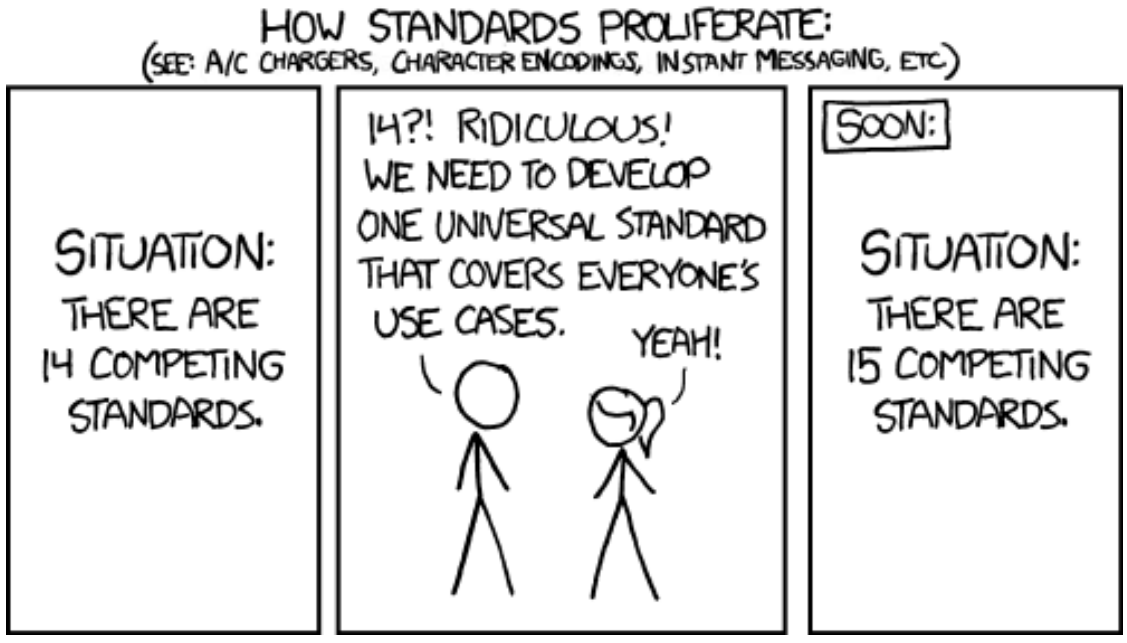
Author's address: 91 Saint Germain Ave., San Francisco, CA 94114.

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

Quelques points d'histoire

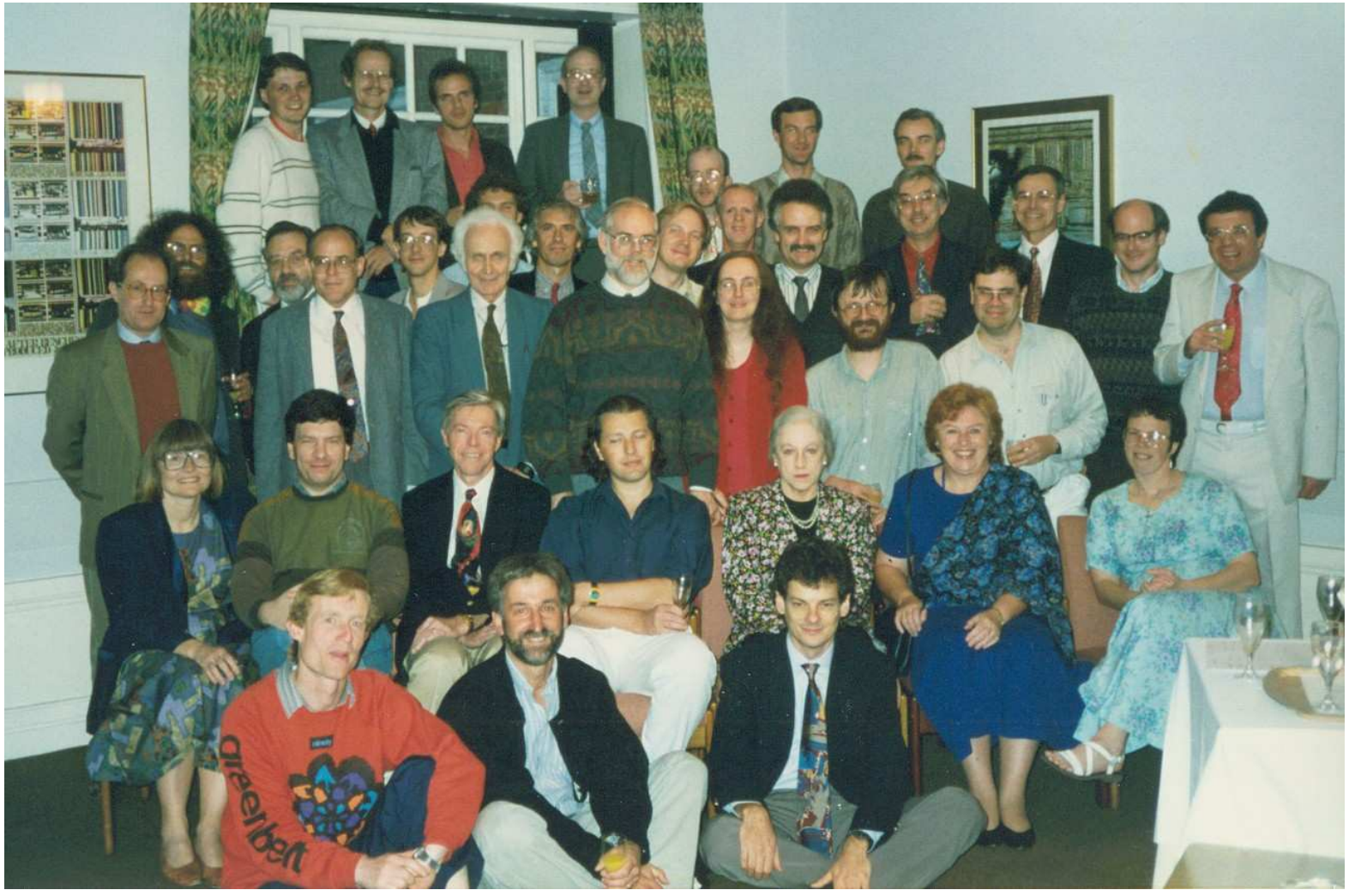
Septembre 1987 : création du *FPLang Committee*



A la fin des années 1980, il existe une dizaine de langages paresseux purement fonctionnels dont *Miranda*, *Lazy ML*, *Orwell*, *Alfl*, *Id*, *Clean*, *Ponder*, *Daisy* ...

Quelques points d'histoire

La situation en Septembre 1987



1987 Functional Programing and Computer Architecture Conference (Portland)

Quelques points d'histoire

Le meeting de Yale (1988)

Les objectifs suivants sont formulés :

1. It should be suitable for teaching, research, and applications, including building large systems.
2. It should be completely described via the publication of a for- mal syntax and semantics.
3. It should be freely available.
4. It should be usable as a basis for further language research.
5. It should be based on ideas that enjoy a wide consensus.
6. It should reduce unnecessary diversity in functional program- ming languages.

Quelques noms envisagés pour le langage :

Semla, Haskell, Vivaldi, Mozart, CFL (Common Functional Language), Funl 88, Semlor, Candle (Common Applicative Notation for Denoting Lambda Expressions), Fun, David, Nice, Light, ML Nouveau (ou Miranda Nouveau), Mirabelle, Concord, LL, Slim, Meet, Leval, Curry, Frege, Peano, Ease, Portland, Haskell B Curry ...

Philosophie du langage Haskell

- Haskell est **paresseux**
  - C'est le point autour duquel est né l'initiative
  - Attention : *paresseux* $\neq$ *non strict*
  - Avantages :
    Permet souvent l'écriture d'un code plus naturel, plus élégant.
    Surcharge en temps est réelle mais constante.
  - Inconvénient :
    Difficulté de raisonner sur la complexité en taille et en espace des programmes

Philosophie du langage Haskell

- Haskell est **paresseux**
  - C'est le point autour duquel est né l'initiative
  - Attention : *paresseux* $\neq$ *non strict*
  - Avantages :
    Permet souvent l'écriture d'un code plus naturel, plus élégant.
    Surcharge en temps est réelle mais constante.
  - Inconvénient :
    Difficulté de raisonner sur la complexité en taille et en espace des programmes

- Haskell est **pur**
  - Le contraire serait incompatible avec l'évaluation paresseuse
  - Le défi de la compatibilité avec les entrées–sorties est remporté grâce à l'utilisation de **monades**

Philosophie du langage Haskell

- Haskell est **paresseux**
  - C'est le point autour duquel est né l'initiative
  - Attention : *paresseux* $\neq$ *non strict*
  - Avantages :
    Permet souvent l'écriture d'un code plus naturel, plus élégant.
    Surcharge en temps est réelle mais constante.
  - Inconvénient :
    Difficulté de raisonner sur la complexité en taille et en espace des programmes

- Haskell est **pur**
  - Le contraire serait incompatible avec l'évaluation paresseuse
  - Le défi de la compatibilité avec les entrées–sorties est remporté grâce à l'utilisation de **monades**

  > *In a call-by-value language, whether functional or not, the temptation to allow unrestricted side effects inside a "function" is almost irresistible. In retrospect, therefore, perhaps the biggest single benefit of laziness is not laziness per se, but rather that laziness kept us pure, and thereby motivated a great deal of productive work on monads and encapsulated state.*

Philosophie du langage Haskell

- Haskell est **paresseux**
  - C'est le point autour duquel est né l'initiative
  - Attention : *paresseux* $\neq$ *non strict*
  - Avantages :
    Permet souvent l'écriture d'un code plus naturel, plus élégant.
    Surcharge en temps est réelle mais constante.
  - Inconvénient :
    Difficulté de raisonner sur la complexité en taille et en espace des programmes

- Haskell est **pur**
  - Le contraire serait incompatible avec l'évaluation paresseuse
  - Le défi de la compatibilité avec les entrées–sorties est remporté grâce à l'utilisation de **monades**

  > *In a call-by-value language, whether functional or not, the temptation to allow unrestricted side effects inside a "function" is almost irresistible. In retrospect, therefore, perhaps the biggest single benefit of laziness is not laziness per se, but rather that laziness kept us pure, and thereby motivated a great deal of productive work on monads and encapsulated state.*

- Haskell implémente les **typeclass**
  - Motivation initiale : surcharge des opérateurs numériques
    Jusque là : différents opérateurs en ML, type numérique unique dans Miranda...
  - Typeclass standards : Eq, Ord, Enum, Show, Read, Foldable, Monoid, Fonctor, Applicative, **Monad**...