# Epigram - View from the left

Bertrand Vidal

18 juin 2009

# Presentation

- functionnal programming language with dependent types
- strong type system that express program specifications
- smooth transition from programming to integrated programs and proofs
- exploits propostions as type principle and pattern matching
- created by C.McBride and J.McKinna
- still implemented by the Epigram group in UK
- available on Linux,Windows and Mac OS X

# Timeline

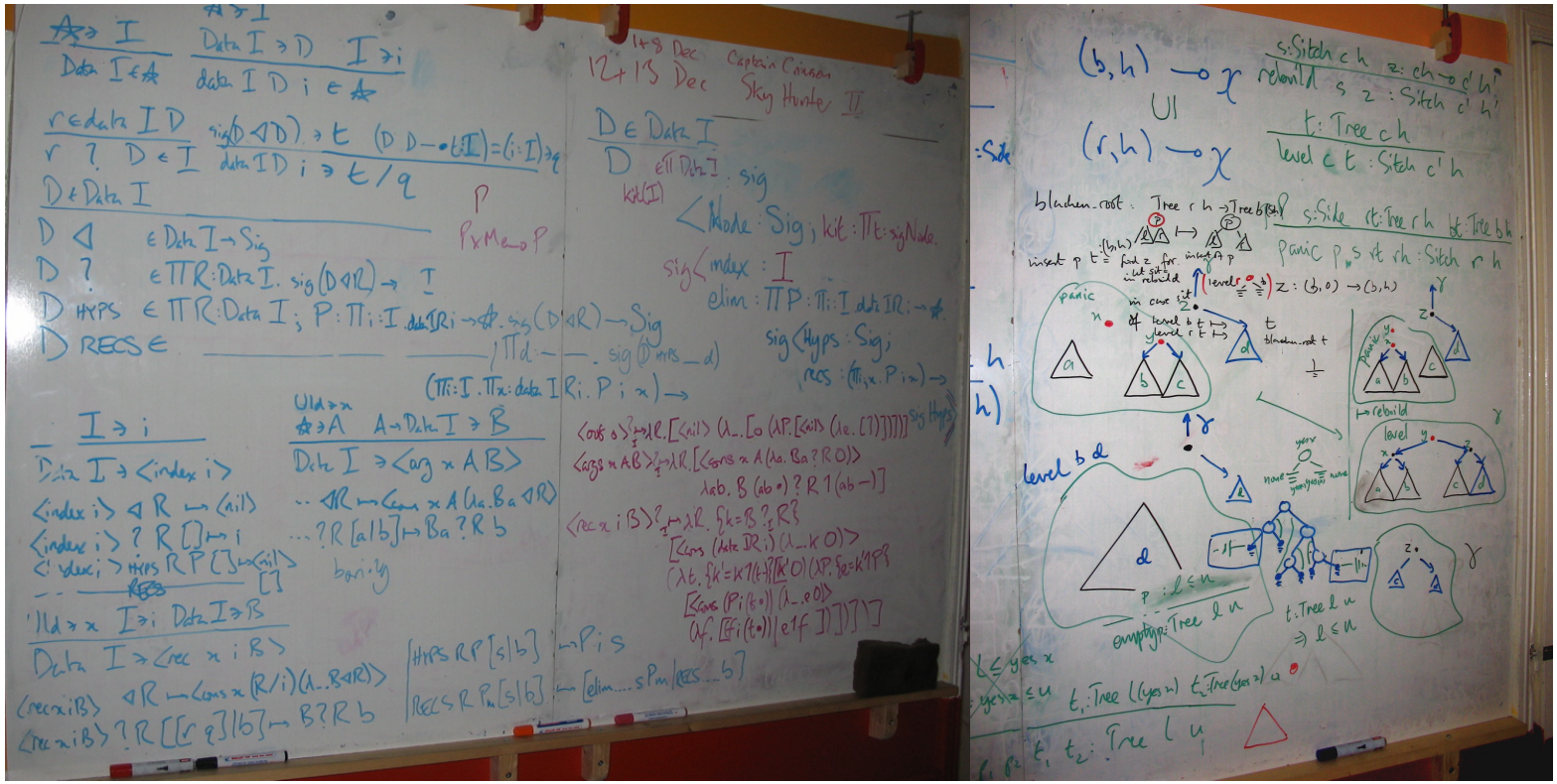
- Late 90s' : Epigram sprout up from the creators minds
- Beginning of the $21^{st}$ century : first release of Epigram
- 20 ? ? : Epigram 2

# Why creating Epigram ?

DTs for proofs $\Longrightarrow$ DTs for programming

# Futur - Epigram 2

Work in progress ⇒ http ://www.e-pig.org/epilogue/



# Creators

## Conor McBride

Lecturer at University of Strathclyde & Alta Systems
Opponent for Ulf Norell's PhD thesis on Agda
contact : conor@strictlypositve.org

# Creators

## James McKinna

Assistant Professor at Radboud University
PhD on proof-carrying code/certified program
contact : james.mckinna@cs.ru.nl

# Contributors

## Thorsten Altenkirch

- Reader (Associate Professor) at the School of Computer Science of the University of Nottingham
- Construction of “elegant and efficient” implementations of type theoretic languages $\Rightarrow$ Epigram

# Contributors

## Healfdene Goguen

- Software engineer at Google, working on large-scale infrastructure
- Co-supervisor of Conor McBride's PhD thesis
- Algorithm for constructor unification for dependent type theories $\Rightarrow$ Epigram

# Contributors

Zhaohui Luo

- Professor of Computer Science - University of London
- Epigram : Innovative Programming via Inductive Families (J.McKinna, C.McBride, P.Callaghan)

# Contributors

James Chapman

- Extraordinary Senior Researcher - Tallinn University of Technology.
- Epigram Reloaded : A Standalone Typechecker for ETT (Thorsten Altenkirch, Conor McBride)

# Contributors

Peter Morris


- Researcher - Department Computer Science University of Nottingham
- Active developer for Epigram

# Contributors

## Edwin Brady

- Researcher - School of Computer Science - University of St Andrews
- Epic : Supercombinator Compiler, intended as a back end for Epigram

# Contributors

## Nicolas Oury

- Marie Curie research fellow in the Foundations of Programming Group of the University of Nottingham.
- Involved in the development of Epigram 2

# Projects using Epigram

- Implemented without a module architecture and huge memory footprint $\Rightarrow$ hard to develop large examples
- Still a "young" programming language
- Active community but relatively small on a very specific topic
- Mainly used for research and publications

# Features

- Pattern matching for types and values based on the data constructors
  - Discriminating tool for DTs
  - Done on the left hand side of intermidiate computations, very powerful with DTs
- Propositions $\Longleftrightarrow$ types (Curry-Howard correspondance)
- Programs $\Longleftrightarrow$ proofs
- Deterministic programs
- Types, as well as constraints, induce program structure $\Rightarrow$ programs as correct by construction

# Grammar

$$
\begin{aligned}
program := \quad & lhs \;\Rightarrow\; expr \;(return\ instruction) \\
| & lhs \;\Leftarrow\; expr\ seq\,[program] \\
| & lhs \mid expr\ [program]
\end{aligned}
$$

$$
\begin{aligned}
decl := \quad & \underline{data}\ sig\ \underline{where}\ sig \\
| & \underline{let}\ sig\ program
\end{aligned}
$$

$$source := seq\,[program]$$

# Inductives families

Class of datatypes aKa Inductives families (Dybjer, 1991)

## Inductive family : **So**

$$\underline{data}\ \frac{b\ :\ \mathbf{Bool}}{\mathbf{So}\ b\ :\ *}$$

- **So** is a collection of types indexed by a boolean value

# Inductives families

Class of datatypes aKa Inductives families (Dybjer, 1991)

## Inductive family : **So**

$$\underline{data}\ \frac{b\ :\ \mathbf{Bool}}{\mathbf{So}\ b\ :\ *}\ \underline{where}\ \frac{}{oh\ :\ \mathbf{So}\ true}$$

- **So** is a collection of types indexed by a boolean value
- **So** true has one element, **So** false has zero element

# Inductives families

Class of datatypes aKa Inductives families (Dybjer, 1991)

## Inductive family : **So**

$$\frac{\Gamma \;\; p : \mathbf{So}\ b,\;\; oh : \mathbf{So}\ true,\;\; p = oh}{\Gamma \;\; ???}$$

- **So** is a collection of types indexed by a boolean value
- **So** true has one element, **So** false has zero element
- assume $p : \mathbf{So}\ b \;\Rightarrow \underline{\mathrm{case}}(p)$

# Inductives families

Class of datatypes aKa Inductives families (Dybjer, 1991)

## Inductive family : **So**

$$\frac{\Gamma\ p : \mathbf{So}\ b,\ \ oh : \mathbf{So}\ \mathit{true},\ \ p = oh}{\Gamma\ p = oh}$$

- **So** is a collection of types indexed by a boolean value
- **So** true has one element, **So** false has zero element
- assume $p : \mathbf{So}\ b \ \Rightarrow \underline{\mathrm{case}}(p)$
  - tells us that $p$ is $oh$

# Inductives families

Class of datatypes aKa Inductives families (Dybjer, 1991)

## Inductive family : **So**

$$\frac{\Gamma\ p : \mathbf{So}\ b,\ \ oh : \mathbf{So}\ true,\ \ p = oh}{\Gamma\ p = oh\ \vdash\ \ b = true}$$

- **So** is a collection of types indexed by a boolean value
- **So** true has one element, **So** false has zero element
- assume $p : \mathbf{So}\ b\ \Rightarrow \underline{\mathrm{case}}(p)$
  - tells us that $p$ is $oh$
  - and "for free" that $b$ must be true

# Inductives families

Class of datatypes aKa Inductives families (Dybjer, 1991)

## Inductive family : **So**

$$\frac{\Gamma\ p : \mathbf{So}\ b,\ \ oh : \mathbf{So}\ true,\ \ p = oh}{\Gamma\ p = oh\ \vdash\ \ b = true}$$

- **So** is a collection of types indexed by a boolean value
- **So** true has one element, **So** false has zero element
- assume $p : \mathbf{So}\ b\ \Rightarrow \underline{\mathrm{case}}(p)$
  - tells us that $p$ is $oh$
  - and "for free" that $b$ must be true
- inspecting $p$ can instantiate $b \Rightarrow$ any type which depends on either

# What is given to the user

You can found on the Epigram home page :

- All required files to run Epigram (libs + examples)
- IDE (xemacs)
- Literature

# How to write an Epigram program

- Start *xemacs*
- Declare data
- Declare functions and let the editor do (most of) the rest
- Compilation and usage

# Demonstration

- Addition for Nat
- Church-Rosser property for combinatory logic

# Questions

## HEEFT U VRAGEN ?

Sources :

- James McKinna himself $\Rightarrow$ THANK YOU !
- http ://www.e-pig.org/
- http ://en.wikipedia.org/wiki/Epigram
- Conor McBride & James McKinna, The view from the left, Departement of Computer Science - University of Durham