Signatures and models for syntax and operational semantics in the presence of variable binding

Ambroise Lafont[1]

[1]DAPI
IMT Atlantique

PhD defense, 2019

Motivation

How do we formally specify a **programming language**?

In the literature: no common well-established discipline.

Differential λ-calculus [Ehrhard-Regnier 2003]

~10 pages (section 2 $\rightarrow$ beginning of section 3) describing the programming language[a] and proving some properties.
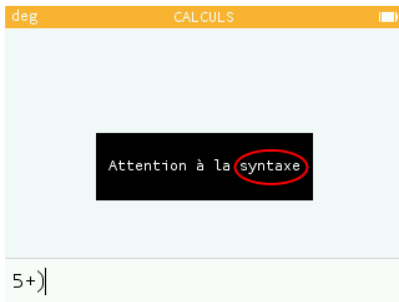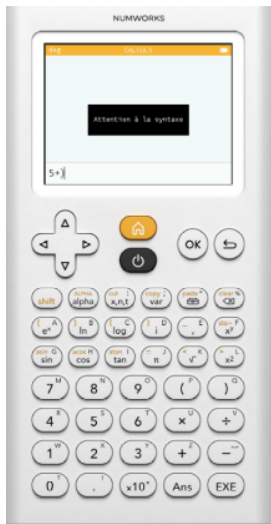
---

[a] not yet satisfyingly addressed by this PhD.

This PhD: a discipline for presenting programming languages

- from small elementary data
- automatically ensuring some properties

What is a programming language?

Example: arithmetic expressions in a calculator

Syntax (of expressions) = formal language

- *vocabulary* : available symbols/keys

- *grammar rules* : what is a valid expression.

e.g. + is a *binary operation*.

Syntax and variables
Focus of this PhD

Variables in expressions

$$(x + 5) \times y$$

$x, y = $ **variables** $= $ placeholders for other expressions
Substitution: variables $\mapsto$ expressions:

$$\begin{cases} \text{replace } x \text{ with } 3 \\ \text{replace } y \text{ with } z \times z \end{cases} \rightsquigarrow \quad (3 + 5) \times (z \times z)$$

Bound variables and α-equivalence

α-**equivalence**:

$$x \mapsto 2 \times x \quad \text{should be identified with} \quad y \mapsto 2 \times y$$

"x is bound by $\mapsto$ in $x \mapsto 2 \times x$"

Ambroise Lafont   Signatures and models for syntax and operational semantics

Syntax and recursion

Recursion (for syntax) = principle for investigating a piece of valid syntactic data.

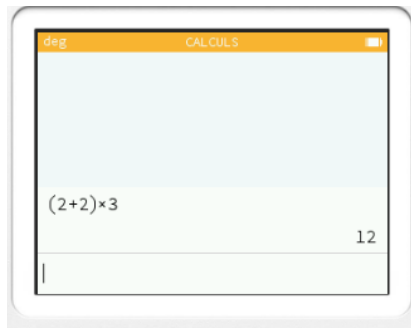### Examples of use of recursion

- count the number of operations in an arithmetic expression
- compute an arithmetic expression

What is a programming language?
Program execution

Program = valid *syntactic* text
Execution = modification of the program:

$$(2 + 2) \times 3 \xrightarrow{\text{1 execution step}} 4 \times 3 \xrightarrow{\text{1 execution step}} 12$$

Operational semantics = description of how programs execute.

Ambroise Lafont   Signatures and models for syntax and operational semantics

What is a programming language?
Finally

Programming language (**PL**) = syntax + operational semantics.

Specification of a PL = features uniquely characterizing a PL.

In 2 steps:

1. syntax

Example: specification of the syntax of arithmetic expressions

- numbers = constants

- + and ×: operations expecting two expressions.

2. semantics

Caveat

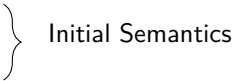There are **ineffective specifications**: no PL satisfies them.

Stupid example

Syntax with two constants 0 and 1 s.t. $0 = 1$ and $0 \neq 1$.

Initial Semantics
Specification through initial semantics for justifying **recursion**.

This PhD:

1. Proposes a mathematical definition of PLs as **reduction monads**;

2. Proposes a notion of **signature** for specifying the syntax and semantics of a reduction monad:
   - **category of models** of a signature
   - initial model = specified object   Initial Semantics
   - initiality $\Rightarrow$ recursion

3. Rules out **ineffective** signatures: identifies a criterion ensuring existence of the initial model.

Outline

Ambroise Lafont   Signatures and models for syntax and operational semantics

Outline

Ingredients

- Programming languages (PLs) as graphs
    - (**Syntax**) vertices = terms
    - (**Semantics**) arrows = reductions between terms
- Simultaneous substitution: variables $\mapsto$ terms
    - monads and modules over them
- (untyped PLs)

Example

λ-calculus with β-reduction:

- **Syntax:** $S, T \quad ::= x \mid S\, T \mid \lambda x.S$

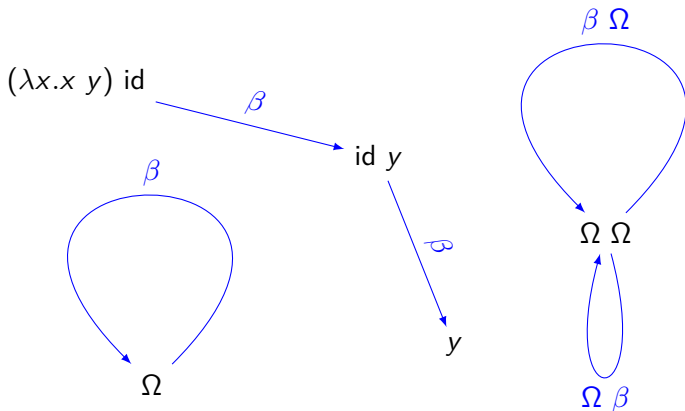- Modulo α-**equivalence**, e.g.

$$\lambda x.x = \lambda y.y$$

- **Reductions:** $(\lambda x.t)\, u \xrightarrow{\beta} t[x := u]$ $+$ congruences

Outline

Ambroise Lafont   Signatures and models for syntax and operational semantics

PLs as graphs
Example: λ-calculus with β-reduction

- (**Syntax**) vertices = terms e.g. $\Omega = (\lambda x.x\,x)(\lambda x.x\,x)$
- (**Semantics**) arrows = reductions

13/48

Graphs
Definition

Graph $=$ a quadruple (A, V, σ, τ) where

$$A = \{\text{arrows}\} \qquad V = \{\text{vertices}\}$$

$$A \underset{\tau}{\overset{\sigma}{\rightrightarrows}} V$$

$$\sigma : \quad A \quad \to V \qquad \tau : \quad A \quad \to V$$
$$t \xrightarrow{r} u \;\mapsto t \qquad\qquad t \xrightarrow{r} u \;\mapsto u$$

$$\sigma(r) \xrightarrow{r} \tau(r)$$

Outline

Ambroise Lafont   Signatures and models for syntax and operational semantics

Simultaneous substitution
Syntax comes with substitution

terms (e.g. λ-terms) = trees with free variables as (distinguished) leaves.

Simultaneous substitution made formal

Free variables indexing

$$X \mapsto \{\text{terms taking free variables in } X\}$$

Example: λ-calculus

$$L(\{x, y\}) = \left\{ \begin{array}{ccccc} \lambda z.z & , & x & , & y & , & x\ y & , & \cdots \\ & & | & & | & & | \quad | & & \\ & & x & & y & & x \quad y & & \end{array} \right\}$$

Simultaneous substitution

$$\forall f : X \to L(Y), \qquad \boxed{\begin{array}{rl} L(X) & \to L(Y) \\ t & \mapsto t[x \mapsto f(x)] \quad (\text{or } t[f]) \end{array}}$$

Monads capture simultaneous substitution

λ-calculus as a monad $(L, \_[\_], \eta)$

1. Simultaneous substitution $(L, \_[\_])$
2. Variables are terms

$$\eta_X : \quad X \quad \rightarrow \quad L(X)$$

$$x \quad \mapsto \quad \underline{x}$$

3. Substitution laws:

$$\underline{x}[f] = f(x) \qquad \qquad t[x \mapsto \underline{x}] = t$$

$+$ associativity:

$$t[f][g] = t\,[x \mapsto f(x)[g]]$$

Substitution for semantics
We saw that syntax is expected to support substitution. This is also true of semantics.

Our notion of PL:

- **Syntax**: a monad $(L, \_[\_], \eta)$
- **Semantics**:

    - graphs $R(X) \underset{\tau_X}{\overset{\sigma_X}{\rightrightarrows}} L(X)$ for each X

    $$R(X) = \begin{array}{l} \text{total set of reductions between} \\ \text{terms taking free variables in } X \end{array}$$

    - substitution of reduction: variables $\mapsto$ L-**terms**.

    $$\frac{t \xrightarrow{r} u}{t[f] \xrightarrow{r[f]} u[f]}$$

Ambroise Lafont   Signatures and models for syntax and operational semantics

Substitution for semantics made formal

R as a module over L

R supports L-monadic substitution:

$$\forall f : X \to \boldsymbol{L}(Y), \qquad \boxed{\begin{aligned} R(X) &\to R(Y) \\ r &\mapsto r[x \mapsto f(x)] \quad \text{(or } r[f]) \end{aligned}}$$

$+$ substitution laws

Remark: any monad T is a module over itself.

σ and τ as L-module morphisms

$$t \xrightarrow{r} u \rightsquigarrow t' \xrightarrow{r[f]} u' \quad \text{with} \quad \begin{cases} t' = t[f] \\ u' = u[f] \end{cases} \quad \text{i.e.,} \quad \begin{cases} \sigma(r[f]) = \sigma(r)[f] \\ \tau(r[f]) = \tau(r)[f] \end{cases}$$

Commutation with substitution $\Leftrightarrow$ Module morphisms $\sigma, \tau : R \to L$.

Reduction monads
Summary: graphs + substitution.

Definition

A **reduction monad** $R \underset{\tau}{\overset{\sigma}{\rightrightarrows}} T$ consists of

- $T =$ monad ($=$ module over itself)
- $R =$ module over T
- $\sigma, \tau : R \to T$ are T-module morphisms.

Example

λ-calculus with β-reduction.

How can we specify a reduction monad?

1. signature for the (syntactic) operations for the monad;

2. reduction rules, involving some specified syntactic operations.

Use of a general notion of **signature** managing this dependency.

Ambroise Lafont   Signatures and models for syntax and operational semantics

Outline

Ambroise Lafont   Signatures and models for syntax and operational semantics

Overview

- Syntax = monad L
- Operations = module morphisms $\Sigma(L) \to L$
- 1-signatures specify operations
- 2-signatures specify operations + equations.

Ambroise Lafont   Signatures and models for syntax and operational semantics

Outline

24/48

Operations as module morphisms

For any model of λ-calculus (in particular for L),

Application commutes with substitution

$$(t\ u)[x \mapsto v_x] = t[x \mapsto v_x]\ u[x \mapsto v_x]$$

Categorical formulation

$L \times L$ supports
$\quad L$-substitution

$\rightsquigarrow$ $L \times L$ is a **module over** L

application commutes
$\quad$ with substitution

$\rightsquigarrow$ $\mathrm{app} : L \times L \to L$ is a

module morphism

[Hirschowitz-Maggesi 2007 : Modules over Monads and Linearity]

Examples of modules

We argued that syntactic operations are **module morphisms**. Basic examples of modules?

Module over a monad T: supports the T-monadic substitution

Examples

- T itself
- $M \times N$ for any modules M and N:

$$\forall (t, u) \in M(X) \times N(X), \qquad X \xrightarrow{f} T(Y),$$

$$\boxed{(t, u)[f] = (t[f], u[f])} \in M(Y) \times N(Y)$$

- $M' =$ **derivative** of a module M:

  X extended with a fresh variable $\diamond$

  $$M'(X) = M(\overbrace{X \amalg \{\diamond\}})$$

used to model an operation binding a variable (Cf next slide).

26/48

Operations as module morphisms
Operations can be combined into a single one.

Operations = module morphisms = maps commuting with substitution:

Example: λ-calculus

$$\begin{aligned}
\text{app}: & \quad L \times L && \to L \\
\text{abs}: & \quad L' && \to L
\end{aligned}
\qquad
\left\{
\begin{aligned}
\text{abs}_X : L(X \amalg \{\diamond\}) &\to L(X) \\
t &\mapsto \lambda \diamond . t
\end{aligned}
\right.$$

Combine operations into a single one:

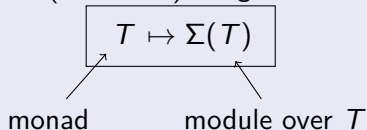$$\boxed{[\text{app}, \text{abs}] : (L \times L) \amalg L' \to L}$$

where (*coproducts* of modules M and N)

$$(M \amalg N)(X) = M(X) \amalg N(X)$$

1-signatures specify operations

Definition

A **1-signature** Σ is a (functorial) assignment

$$T \mapsto \Sigma(T)$$

monad   module over T

Definition (model of a 1-signature Σ)

A **model** of Σ is a pair (T, m) denoted by $\Sigma(T) \xrightarrow{m} T$ s.t.

- T is a monad
- $\Sigma(T) \xrightarrow{m} T$ is a T-module morphism

Example: λ-calculus

$$[\text{app}, \text{abs}] : \Sigma_{LC}(L) \to L \qquad \text{where } \Sigma_{LC}(L) = (L \times L) \amalg L'$$

Ambroise Lafont   Signatures and models for syntax and operational semantics

Syntax
We defined 1-signatures and their models. When is a signature effective?

(suitable notion of model morphism [Hirschowitz-Maggesi 2012])

Definition

The **syntax** specified by a 1-signature Σ is the initial object in its category of models.

Question: Does the syntax exist for every 1-signature?

Answer: No.

Counter-example: $\Sigma(R) = \mathcal{P} \circ R$

Powerset endofunctor on *Set*.

Ambroise Lafont   Signatures and models for syntax and operational semantics

Initial semantics for algebraic 1-signatures
We gave examples of effective 1-signatures. They were all **algebraic**.

Definition

Algebraic 1-signatures $=$ 1-signatures built out of derivatives, finite products, disjoint unions, and the 1-signature $\Theta : T \mapsto T$.

Algebraic 1-signatures $\simeq$ binding signatures [Fiore-Plotkin-Turi 1999]
$\Rightarrow$ specification of n-ary operations, possibly binding variables.

Theorem (Fiore-Plotkin-Turi 1999)

Syntax exists for any algebraic 1-signature.

Example

λ-calculus

Question: Specify syntactic operations subject to some equations?

e.g. a *commutative* or *associative* binary operation.

Quotient of algebraic signatures

We saw that algebraic signatures are effective. Can we specify effectively operations subject to equations?

Theorem

Syntax exists for any "quotient" of algebraic 1-signatures.

Example

a *commutative* binary operation $+$:

$$\forall a, b, \quad a + b = b + a$$

Ambroise Lafont Signatures and models for syntax and operational semantics

Outline

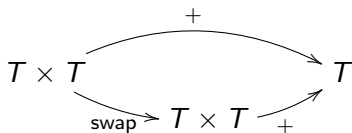Ambroise Lafont   Signatures and models for syntax and operational semantics

Example: a commutative binary operation

Specification of a binary operation

1-signature	$T \mapsto T \times T$
model	$\begin{array}{c} T \times T \\ \downarrow + \\ T \end{array}$

Question What is an appropriate notion of model for a
commutative binary operation?

- a monad T
- with a binary operation   } a model $T \times T \xrightarrow{+} T$ of $\Theta \times \Theta$
- s.t.

$$T \times T \xrightarrow{\quad + \quad} T \quad\quad \text{where} \quad \text{swap}(t, u) = (u, t)$$

with the lower path $\text{swap} \to T \times T \to_{+}$

Equations

$\Sigma = 1$-signature (e.g. binary operation $\Sigma(T) = T \times T$)

Definition

A Σ-**equation** $A \underset{v}{\overset{u}{\rightrightarrows}} B$ is a (functorial) assignment

$$\underbrace{M = (\Sigma(T) \to T)}_{\text{model of } \Sigma} \quad \mapsto \quad \left(A(M) \underset{v_M}{\overset{u_M}{\rightrightarrows}} B(M) \right)$$

model of Σ $\qquad$ parallel pair of T-module morphisms

Example (Binary commutative operation)

$$\Sigma(T) = T \times T \qquad \begin{array}{c} T \times T \\ \downarrow + \\ T \end{array} \quad \mapsto \quad T \times T \underset{\text{swap}}{\overset{+}{\longrightarrow}} T \times T \overset{+}{\longrightarrow} T$$

Ambroise Lafont   Signatures and models for syntax and operational semantics
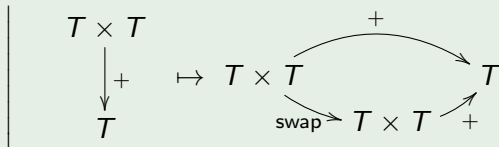
2-signatures and their models

We defined equations. A set of equations yields a 2-signature.

Definition

A **2-signature** is a pair (Σ, E) where

- Σ is a 1-signature for monads
- E is a set of Σ-equations

Definition

A **model** of a 2-signature (Σ, E) consists of:

- a model $M = \begin{pmatrix} \Sigma(T) \\ \big\downarrow \\ T \end{pmatrix}$ of Σ s.t.

$$\forall\, A \underset{v}{\overset{u}{\rightrightarrows}} B \,\in E, \quad \boxed{u_M = v_M} : A(M) \to B(M)$$

morphism of models = morphisms as models of Σ.

Initial semantics for algebraic 2-signatures

We defined 2-signatures and their models. When is a 2-signature effective?

Theorem

*Any **algebraic** 2-signature has an initial model.*

Definition

A 2-signature (Σ, E) is **algebraic** if:

- Σ is algebraic

- E consists of **elementary** Σ-equations

Main instances of elementary Σ-equations

$$A \rightrightarrows B \text{ s.t.} \qquad A \begin{pmatrix} \Sigma(T) \\ \Downarrow \\ T \end{pmatrix} = \Phi(T) \qquad B \begin{pmatrix} \Sigma(T) \\ \Downarrow \\ T \end{pmatrix} = T$$

for some *algebraic* 1-signature Φ.

(e.g. $\Phi(T) = T \times T$ for commutativity)

Ambroise Lafont　　Signatures and models for syntax and operational semantics

Examples of elementary equations

We saw that elementary Σ-equations yield effective 2-signatures.

Example

- *associativity* of a binary operation
- fixpoint equation

$$(\text{fix}\, f := t) \quad = \quad t[f := (\text{fix}\, f := t)]$$

- β-reduction as an equation:

$$(\lambda x.t)\, u = t[x := u]$$

What if we want β-reduction as a *reduction* rather than an *equation*?

Ambroise Lafont   Signatures and models for syntax and operational semantics

Outline

Specifying reduction monads

λ-calculus with (small-step) β-reduction as a reduction monad:
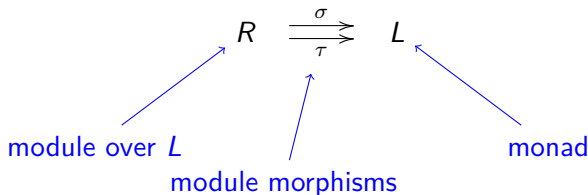
- vertices $= L =$ initial model of the signature of λ-calculus.
- arrows $= R, \sigma, \tau = ?$
    - specified through *reduction rules* (to be made formal):

$$(\lambda x.t)\, u \to t[x := u] \qquad \frac{t \to t'}{t\, u \to t'\, u} \qquad \dots$$

Outline

1. Reduction monads
   - Graphs
   - Substitution

2. Syntax
   - Operations
   - Equations

3. **Semantics**
   - **Reduction rules**
   - Reduction signatures

Analysis of a reduction rule

Example: binary congruence for application.

metavariables: as a L-module L^4

$$\overbrace{t, t', u, u'} \qquad \mapsto$$

hypotheses

$$\frac{t \to t' \qquad u \to u'}{t\,u \to t'\,u'}$$

conclusion

Hypothesis/conclusion = pair of λ-terms using metavariables

- as parallel module morphisms $L^4 \rightrightarrows L$

$$\text{e.g.,} \quad t\,u \to t'\,u' : \qquad (t, t', u, u') \mapsto t\,u$$
$$(t, t', u, u') \mapsto t'\,u'$$

- **Generalization**: $L \rightsquigarrow$ any model $\Sigma_{LC}(T) \to T$ of Σ_{LC}:

$$\text{(application denoted by app} : T \times T \to T)$$
$$\text{e.g.,} \quad t\,u \to t'\,u' : \qquad (t, t', u, u') \mapsto \mathsf{app}(t, u)$$
$$(t, t', u, u') \mapsto \mathsf{app}(t', u')$$

42/48

Reduction rules
Definition

Let Σ = signature for monads (e.g. Σ_{LC} for congruence for application).

Definition of Σ-reduction rules

A Σ-**reduction rule** $(\vec{\sigma}, \vec{\tau})$

$$\frac{\sigma_1 \to \tau_1 \quad \ldots \quad \sigma_n \to \tau_n}{\sigma_0 \to \tau_0}$$

assigns (functorially) to each model $\Sigma(T) \to T$:

- $V(T) = T$-module of metavariables (e.g. $V(T) = T^4$)

- parallel T-module morphisms $V(T) \underset{\tau_{i,T}}{\overset{\sigma_{i,T}}{\rightrightarrows}} T'^{\cdots'}$

We write
$$\sigma_i, \tau_i : V \to \Theta^{(n_i)} \qquad n_i = \text{number of derivatives}$$

43/48

Outline

Reduction signatures

Definition

A **reduction signature** is a pair $(\Sigma, \mathfrak{R})$ where

- Σ is a signature for monads (1- or 2-signature)
- $\mathfrak{R}$ is a family of Σ-reduction rules

Example: λ-calculus with β-reduction

- $\Sigma = \Sigma_{LC}$
- Σ-reduction rules:
    - β-reduction
    - congruence for application and **abstraction** ($T' \xrightarrow{\text{abs}} T$):

$$\frac{u \to u'}{\lambda x.u \to \lambda x.u'} \rightsquigarrow \frac{\pi_1 \to \pi_2}{\text{abs} \circ \pi_1 \to \text{abs} \circ \pi_2} \qquad \frac{T' \times T' \begin{smallmatrix} \pi_{1,T} \\ \rightrightarrows \\ \pi_{2,T} \end{smallmatrix} T'}{T' \times T' \begin{smallmatrix} \text{abs} \circ \pi_{1,T} \\ \rightrightarrows \\ \text{abs} \circ \pi_{2,T} \end{smallmatrix} T}$$

Ambroise Lafont   Signatures and models for syntax and operational semantics

Models
We defined **reduction signatures**. What are their models?

A **model** of a signature $(\Sigma, \mathfrak{R})$ consists of:

- a reduction monad $R \underset{\tau}{\overset{\sigma}{\Longrightarrow}} T$ with a Σ-model structure on T

- for each reduction rule

$$\boxed{\frac{\sigma_1 \to \tau_1 \quad \dots \quad \sigma_n \to \tau_n}{\sigma_0 \to \tau_0} op} \quad V \underset{\tau_i}{\overset{\sigma_i}{\Longrightarrow}} \Theta^{(n_i)} \quad \text{in } \mathfrak{R},$$

  - a mapping, for each $v \in V(T)(X)$,

  $$\begin{pmatrix} \sigma_1(v) \xrightarrow{r_1} \tau_1(v) \\ \dots \\ \sigma_n(v) \xrightarrow{r_n} \tau_n(v) \end{pmatrix} \quad \mapsto \quad \sigma_0(v) \xrightarrow{op(r_1,\dots r_n)} \tau_0(v)$$

  - compatible with substitution:

  $$op(r_1, \dots r_n)[f] = op(r_1[f], \dots, r_n[f])$$

46/48

Initiality
We defined **models** of a **reduction signature**. When is a signature effective?

(suitable notion of model morphism)

Theorem

Σ has an initial model (e.g. Σ is algebraic) $\Rightarrow$ $(\Sigma, \mathfrak{R})$ has an initial model.

Examples

- λ-calculus with small-step β-reduction
- λ-calculus with explicit substitution [Kesner 2009].

  A Theory of Explicit Substitutions with Safe and Full Composition

Generalizing from *graphs* to *bipartite graphs* yields more examples:

Examples

- (small-step or big-step) cbv λ-calculus.
- π-calculus

Conclusion

Summary

- PLs as reduction monads
- Signatures for reduction monads with effectivity theorem

Articles

- AHLM CSL 2018 about *quotients of algebraic 1-signatures*
- AHLM FSCD 2019 about *algebraic 2-signatures*
- AHLM POPL 2020 about *reduction monads*
- HHL (on HAL): extension to bipartite graphs and simply typed PLs

AHLM = Ahrens, A. Hirschowitz, *Lafont*, Maggesi
HHL = A. Hirschowitz, T. Hirschowitz, *Lafont*