GRAMMAIRES ATTRIBUÉES ET FOLDS : OPÉRATEURS DE CONTRÔLE GÉNÉRIQUES

Etienne Duris, Didier Parigot
Gilles Roussel, Martin Jourdan

E-mail : `Prenom.Nom@inria.fr`

`http://www-rocq.inria.fr/oscar`

Journées Francophones des Langages Applicatifs

Dolomieu, Isère - 20, 21 Janvier 1997

OBJECTIF...

Comparaison de transformations de programmes
en PROGRAMMATION FONCTIONNELLE et
en GRAMMAIRES ATTRIBUÉES

- programmation dirigée par la structure
- élimination des structures intermédiaires : déforestation
- pouvoir d'expression, paradigmes...

AUTOUR DES FOLDS...

- 1977 - Burstall & Darlington
- 1988 - Bird & Wadler; (Algorithme de déforestation)
- 1991 - Catamorphismes, etc... Meijer *et al* (bananas)
- 1993 - Implantations (Haskell) : foldr/buildr (Gill & Launchbury & Peyton Jones)
  1993 - Instanciations : fold - normalisation (Sheard & Fegaras)
- 1995 - Hylo-Fusion : hylos en triples (Takano & Meijer)
- 1996 - Déf. récursives → Hylomorphismes (Hu & Iwasaki & Takeishi)
- 1996 - Fonctions polytypiques (Meertens...)

• Structure des données plus explicite

• Opérateur générique

→ **fusion, optimisations**

GRAMMAIRES ATTRIBUÉES

- Introduites par Knuth en 1968
  - Spécifications déclaratives, dirigées par la structure
  - Analyses statiques fines (durées de vie)
  - Transformations (composition descriptionnelle)
  - Généricité, modularité
- Grammaires Attribuées Dynamiques en 1995-96

→ **langage fonctionnel du premier ordre**

PLAN

- Cas simple :
  - fold *versus* grammaire attribuée purement synthétisée
  - DÉFORESTATION :
    algorithme de normalisation des folds du premier ordre
    versus
    composition descriptionnelle des GA purement synthétisées
- Cas général :
  - fold d'ordre supérieur *versus* GA générale (attributs hérités)
    et leur déforestation. . .

    (Sheard93 et autres. . .)

STRUCTURES et TYPES

- FOLDS

types algébriques, constructeurs, variables de type

$$
\begin{array}{lcl}
\mathsf{list}(\alpha) & = & \mathsf{Nil} \mid \mathsf{Cons}(\alpha,\mathsf{list}(\alpha)) \\
\mathsf{tree}(\alpha) & = & \mathsf{Tip}(\alpha) \mid \mathsf{Node}(\mathsf{tree}(\alpha),\mathsf{tree}(\alpha)) \\
\mathsf{int} & = & \mathsf{Zero} \mid \mathsf{Succ}(\mathsf{int})
\end{array}
$$

- GRAMMAIRES ATTRIBUÉES

syntaxe, productions, non-terminaux

Type list :
Cons $\rightarrow \alpha$ list
Nil $\rightarrow$

Type tree :
Node $\rightarrow$ tree tree
Tip $\rightarrow \alpha$

Type int :
Succ $\rightarrow$ int
Zero $\rightarrow$

EXEMPLE (FOLD)

- Longueur d'une liste

$$\text{length}(x) = \text{fold}^{list}\ (\quad \begin{array}{l} \lambda().\text{Zero}\ , \\ \lambda(a,r).\text{Succ}(r)\ )\ x \end{array}$$

- Définition statique de fold^{list} à partir des foncteurs du type list :

$$\begin{array}{lcl} \text{fold}^{list}\ (f_n, f_c)\ \text{Nil} & = & f_n() \\ \text{fold}^{list}\ (f_n, f_c)\ \text{Cons}(a,l) & = & f_c(a, \text{fold}^{list}\ (f_n, f_c)\ l) \end{array}$$

EXEMPLE (GA)

- Longueur :

(notation classique)

`Nil ->`

$s_{\uparrow \texttt{Nil}} = \mathsf{Zero}$

`Cons -> a list`

$s_{\uparrow \texttt{Cons}} = \mathsf{Succ}(s_{\uparrow \texttt{list}})$

(notation “lambda”)

`Nil ->`

$s_{\uparrow \texttt{Nil}} = (\lambda().\mathsf{Zero})()$

`Cons -> a list`

$s_{\uparrow \texttt{Cons}} = (\lambda(a,r).\mathsf{Succ}(r))(a, s_{\uparrow \texttt{list}})$

- DIFFÉRENTES CLASSES DE GRAMMAIRES ATTRIBUÉES

(Règles sémantiques → système d'équations → évaluateur d'attributs)

FOLD ÉQUIVALENT À GA S^1

- Représentation d'un programme avec un fold :

$$\text{fold}^{list}\ (f_n, f_c)$$

- Le même programme exprimé avec une GA :

$$
\begin{array}{l}
\texttt{Nil ->} \\
\qquad s_{\uparrow\texttt{Nil}} = f_n \\
\texttt{Cons -> a list} \\
\qquad s_{\uparrow\texttt{Cons}} = f_c(\mathsf{a}, s_{\uparrow\texttt{list}})
\end{array}
$$

- La GA est purement synthétisée (S^1).

- L'opérateur fold^{list} défini à partir des foncteurs constitue un évaluateur d'attributs correct pour la GA S^1.

- Paramètres : implicites en fold / explicites en GA.

THÉORÈME DE PROMOTION DU FOLD

- Théorie algébrique, morphismes, catégories...
  (free theorem, fusion law...)

→ pour le type liste :

$$\frac{\begin{array}{lcl} \phi_n \ () & = & \mathsf{g}\ (f_n()) \\ \phi_c\ (\mathsf{a}\ ,\ \mathsf{g}(\mathsf{r})) & = & \mathsf{g}\ (f_c(\mathsf{a},\mathsf{r})) \end{array}}{\mathsf{g}\ (\mathsf{fold}^{list}(f_n, f_c)\ \mathsf{x}) = \mathsf{fold}^{list}(\phi_n, \phi_c)\mathsf{x}}$$

La composition d'une fonction avec un fold est un fold

- Algorithme de Normalisation :
  - Généralisation
  - Application à une Construction
  - Promotion

EXEMPLE : *length(append)* (1)

- Longueur d'une liste :

$$\text{length}(x) = \text{fold}^{list} \; (\quad \lambda().\text{Zero} \;, \quad \lambda(a,r).\text{Succ}(r) \;) \; x$$

Concaténation de deux listes :

$$\text{append}(x,y) = \text{fold}^{list} \; (f_n, f_c) \; x$$

où $f_n = \lambda().y$
$f_c = \lambda(a,r).\text{Cons}(a,r)$

- Théorème de Promotion du Fold :

$$\text{length}(\text{append}(x,y)) = \text{fold}^{list} \; (\phi_n \;,\; \phi_c) \; x$$

où $\phi_n() = \text{length}(f_n())$
$\phi_c(r_1,r_2) = \text{length}(f_c(x_1,x_2))$

avec $[x_1/r_1, \text{length}(x_2)/r_2]$

EXEMPLE : *length(append)* (2)

- Algorithme de Normalisation :

$\phi_n()$ = length(y)

$\phi_c(r_1,r_2)$ = length(Cons(x_1,x_2)) où [x_1/r_1, length(x_2)/r_2]

(Application à une Construction)

$\phi_c(r_1,r_2)$ = Succ(length(x_2))

(Généralisation de length(x_2) par r_2)

$\phi_c(r_1,r_2)$ = Succ(r_2)

- Finalement : aucun Cons dans length(append) normalisé

fold^{list}($\lambda()$. length(y),
$\lambda(r_1,r_2)$.Succ(r_2)) x

LA COMPOSITION DESCRIPTIONNELLE

- Ganzinger & Giegerich (1984)

- Idée : Projection des règles sémantiques
  indépendante de tout ordre d'évaluation

$$
\begin{tikzcd}
 & \Delta \arrow[dr, "\delta"] & \\
\Gamma \arrow[ur, "\gamma"] \arrow[rr, "\delta \circ \gamma"'] & & \Omega
\end{tikzcd}
$$

Dans γ : $\quad \mathtt{C}_\Gamma$ `->` ...

$$a_{\uparrow \mathtt{C}_\Gamma} = C_\Delta(\ldots)$$

Dans δ : $\quad \mathtt{C}_\Delta$ `->` ...

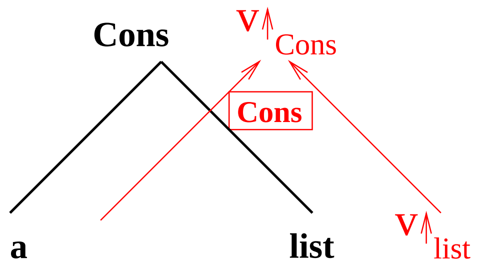
$$b_{\uparrow \mathtt{C}_\Delta} = f(\ldots)$$

Dans $\delta \circ \gamma$: $\quad \mathtt{C}_\Gamma$ `->` ...

$$ab_{\uparrow \mathtt{C}_\Gamma} = f(\ldots)$$

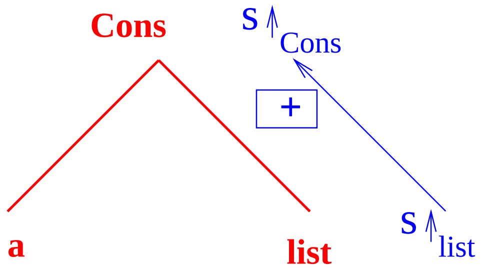
EXEMPLE : *length(append)*

append :



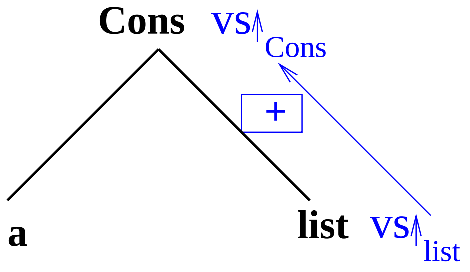
$$v_{\uparrow\mathtt{Cons}} = \mathrm{Cons}(a, v_{\uparrow\mathtt{list}})$$

length :



$$s_{\uparrow\mathtt{Cons}} = \mathrm{Succ}(s_{\uparrow\mathtt{list}})$$

length ∘ append :



$$vs_{\uparrow\mathtt{Cons}} = \mathrm{Succ}(vs_{\uparrow\mathtt{list}})$$

ALGORITHME de NORMALISATION vs COMPOSITION DESCRIPTIONNELLE

- Théorème de Promotion $\Leftrightarrow$ Correction de la Composition Descriptionnelle
- Résultats de déforestation identiques pour fold de 1^{er} ordre et GA S^1
- Théorème de Promotion lié aux foncteurs (méthode d'évaluation)
  Composition descriptionnelle indépendante de la méthode d'évaluation
- Schéma de récursion du calcul d'un fold isomorphe à la structure
  (pas nécessairement en GA)

ÉVALUATION PARTIELLE GÉNÉRALISÉE

vs

TRANSFORMATION PUREMENT SYMBOLIQUE

CAS GÉNÉRAL

- Fold du premier ordre non normalisable

- Inversion de liste en premier ordre

$$\begin{aligned}\text{reverse}(x) = \text{fold}^{list}(\quad & \lambda().\text{Nil}, \\ & \lambda(a,r).\text{append}(r,\text{Cons}(a,\text{Nil}))\ )\ x\end{aligned}$$

- en second ordre

$$\begin{aligned}\text{reverse}(x) = \text{fold}^{list}(\quad & \lambda().\lambda(w).w, \\ & \lambda(a,r).\lambda(w).r(\text{Cons}(a,w))\ )\ x\ \text{Nil}\end{aligned}$$

- en grammaire attribuée (attribut hérité)

Nil ->

$s_{\uparrow\texttt{Nil}} = h_{\downarrow\texttt{Nil}}$

Cons -> a list

$s_{\uparrow\texttt{Cons}} = s_{\uparrow\texttt{list}}$

$h_{\downarrow\texttt{list}} = \text{Cons}(a, h_{\downarrow\texttt{Cons}})$

DÉFORESTATION : (reverse $\circ$ reverse)

En FOLD : Algorithme de Normalisation

- en utilisant des formes de type particulières (Sheard93)

  zéro-constructeurs (pas tous les types)

  $\longrightarrow$ reverse $\circ$ reverse = copie (premier ordre)

- avec types généraux, et système de réécriture (Fegaras94)

  plus de transformations particulières, mais des exceptions

  $\longrightarrow$ reverse $\circ$ reverse reste au second ordre

- définitions récursives $\rightarrow$ hylo-fusion (Launchbury, Takano, Hu 95)

  de plus en plus général, de plus en plus automatique, mais. . .

  $\longrightarrow$ reverse $\circ$ reverse reste au second ordre

DÉFORESTATION : (reverse $\circ$ reverse)

En GA : Composition Descriptionnelle

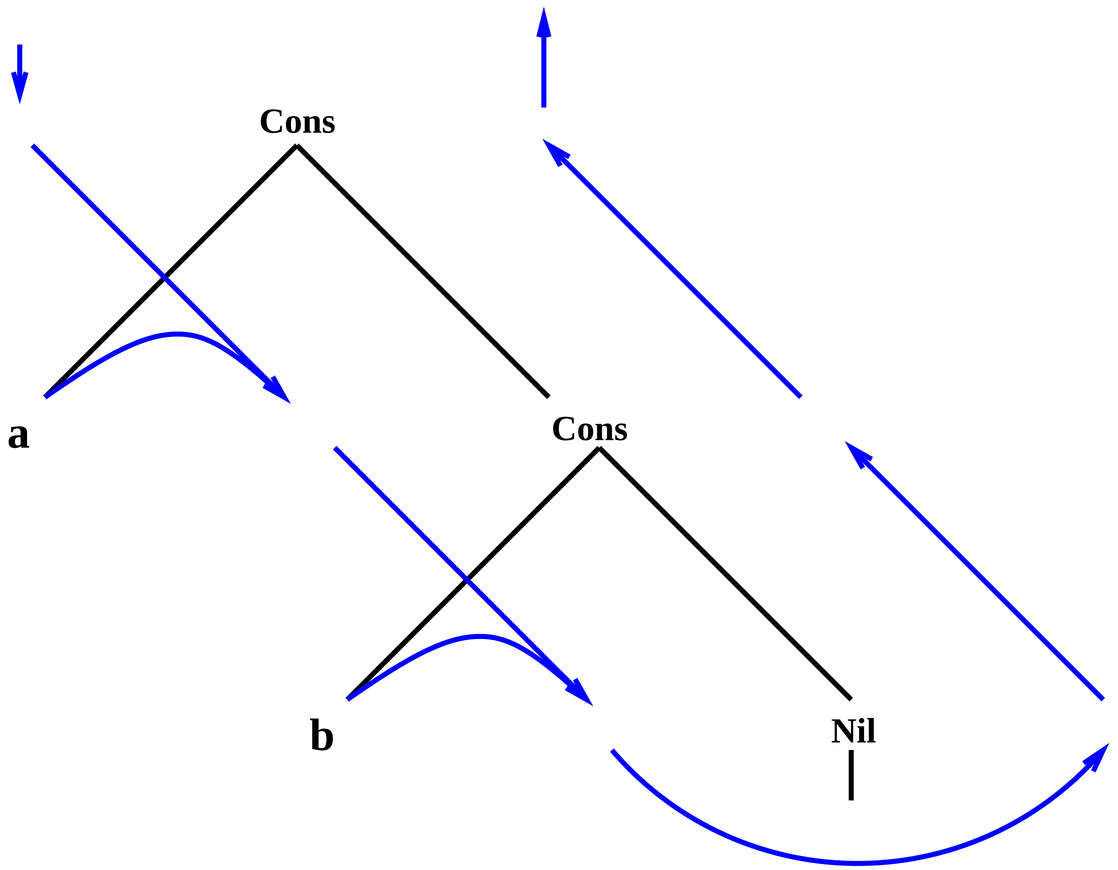
- Le formalisme et l'algorithme sont les mêmes que pour les GA S[1]

  reverse $\circ$ reverse est une grammaire attribuée "naturelle" (attr. hérités) avec des règles de *propagation* des attirbuts.

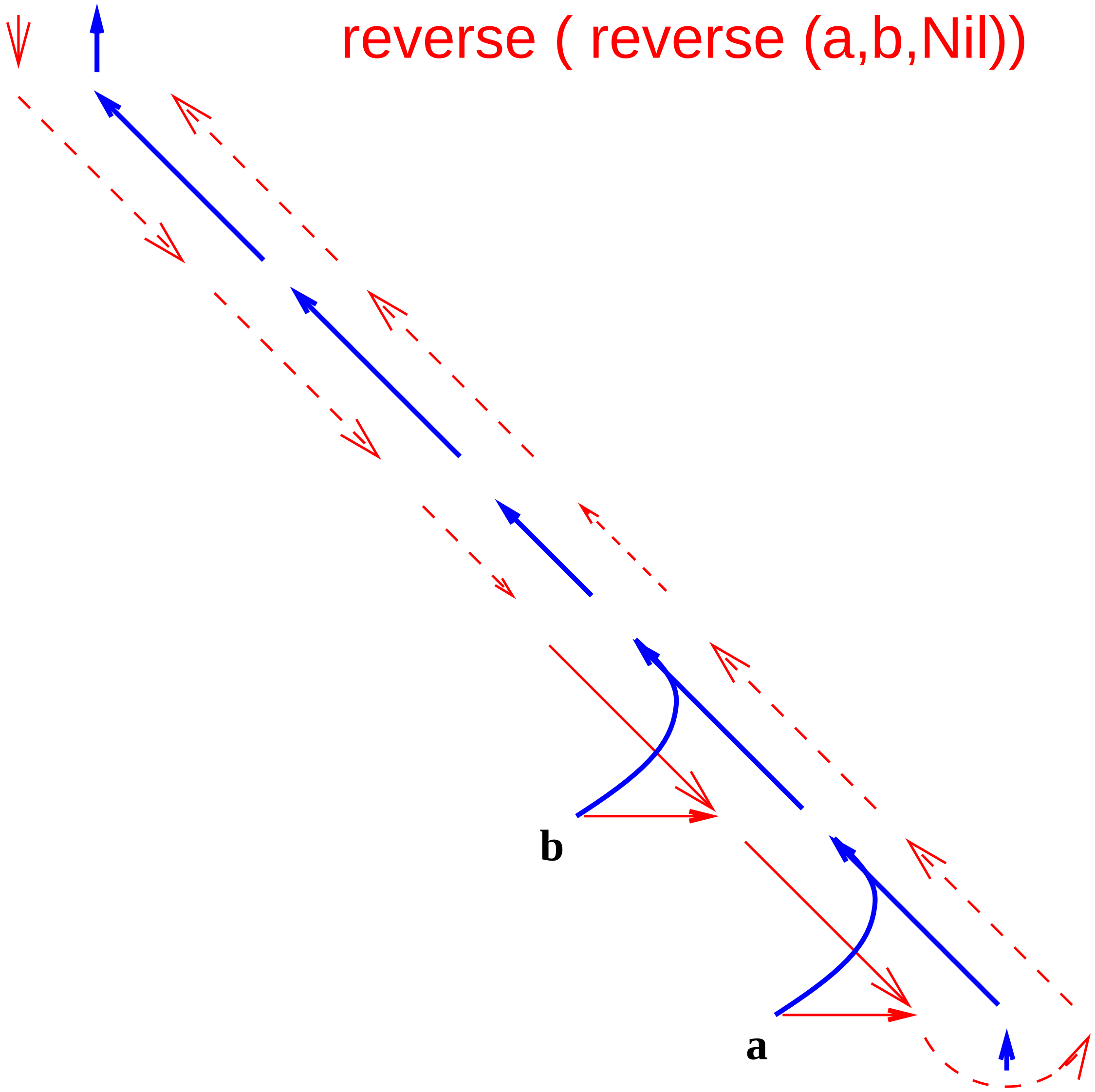
- élimination des règles de copies

$$\longrightarrow \text{reverse} \circ \text{reverse} = \text{copie}$$

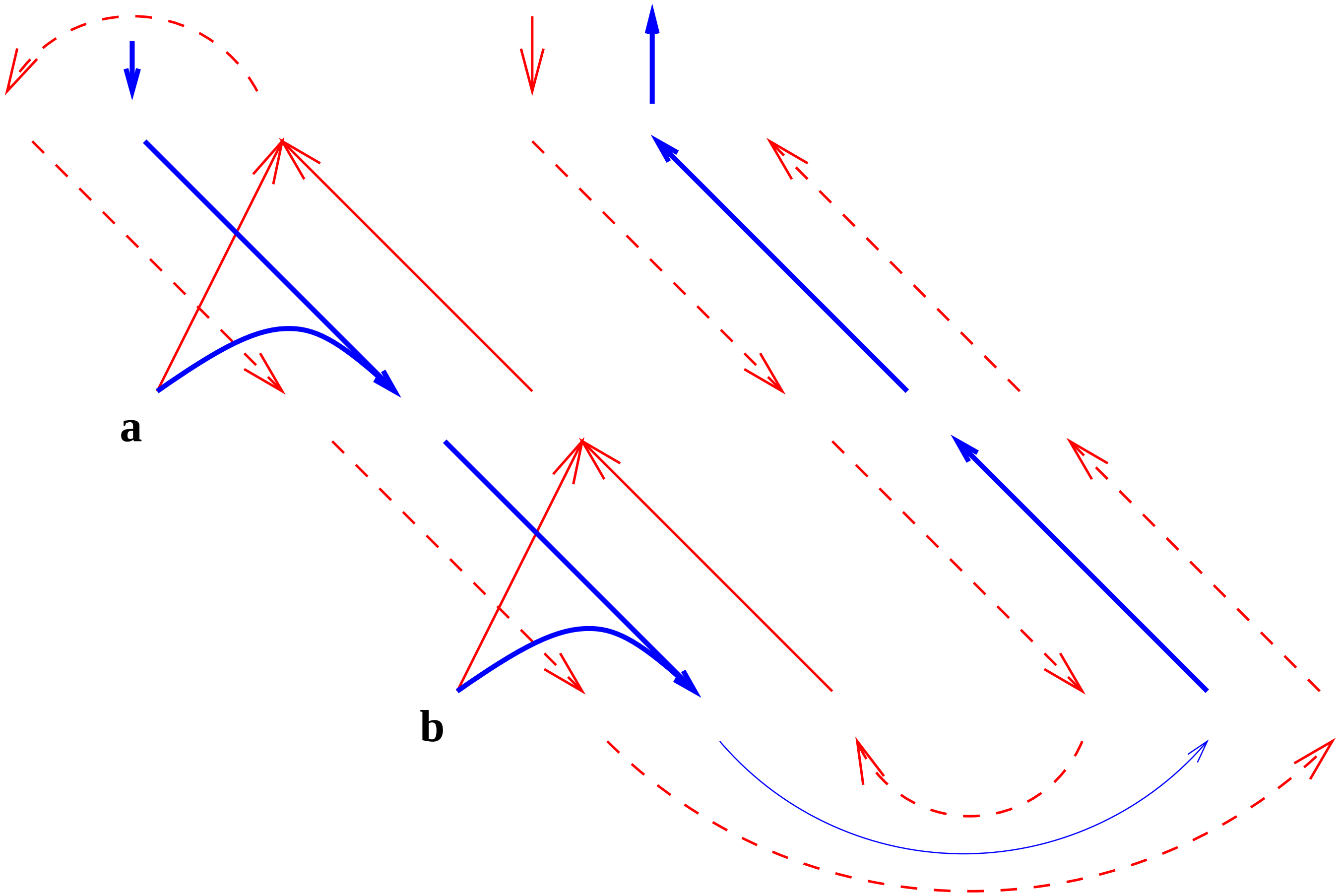
reverse (a,b,Nil)



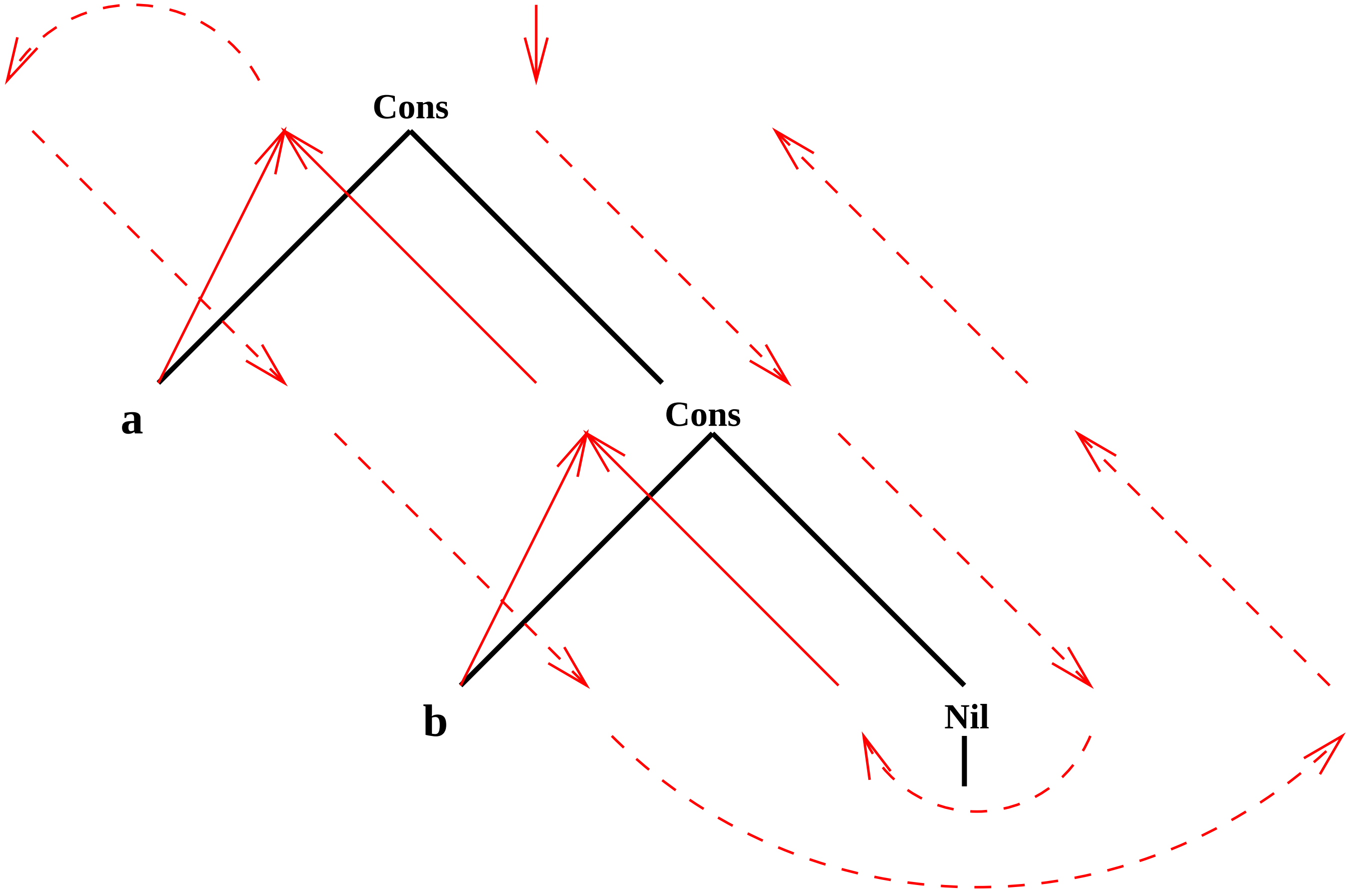
reverse (reverse (a,b,Nil))



reverse (reverse (a,b,Nil))



(reverse ∘ reverse) (a,b,Nil)



CONCLUSION

- Folds du premier ordre $\equiv$ GA S[1]

  Formalismes très proches (méthode d'évaluation nécessaire ou non)

  Algorithme de Normalisation $\equiv$ Composition Descriptionnelle.

  $\longrightarrow$ Évaluation partielle généralisée / transformation symbolique

- Ordre supérieur $\Leftrightarrow$ attributs hérités ?

  Approches différentes (historique), ressemblances de résultats...

  Fertilisation croisée ? Conditions aux limites transposables ?

  $\longrightarrow$ ordre supérieur dans le résultat / formalisme de base naturel (hérités)

  Nécessité du foncteur / méthode d'évaluation indifférente

- "Bonnes" propriétés intrinsèques à la notation GA.

  Attributs hérités représentent l'ordre supérieur.

  Spécification dirigée par la structure (localement).

TRAVAUX FUTURS

- Déforestation, optimisation, fusion
- Formalisation : cata-, ana-, hylo- morphismes...
- Généricité, modularité, instanciation (Composition de GA)
  - Farrow 92
  - Le Bellec *et al.* 93
  - Correnson