# Unification, Réécriture

Laura Brillon
laura.brillon[at]math.univ-toulouse.fr

Centre Universitaire Jean-François Champollion

Octobre 2015

# Sommaire

# Induction sur les nombres naturels

**Question typique :** On veut montrer que

$$\forall n \in \mathbb{N}, \mathcal{P}(n)$$

**Démarche :**

- Montrer $\mathcal{P}(0)$
- Montrer $\mathcal{P}(n) \Rightarrow \mathcal{P}(n+1)$

**Exercice :**

Montrer

$$\forall n \in \mathbb{N}, \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

# Termes

Soient $\mathcal{L}$ un ensemble de symboles de *fonctions d'arité donnée* et un ensemble infini $\mathcal{V}$ de variables.

## Définition

L'ensemble $\mathcal{T}$ des **termes** sur $\mathcal{L}$ est défini inductivement par :

- $\mathcal{V} \subset \mathcal{L}$ (les variables sont contenues dans les termes)
- Si $f$ est un symbole d'arité $k$ et si $t_1, ..t_k \in \mathcal{T}$, alors $f(t_1, .., t_k) \in \mathcal{T}$.

· Les fonctions d'arité 0 sont appelées les **constantes**.

· Si $t$ est un terme, on note $t[x_1, .., x_n]$ si seules les variables $x_1, .., x_n$ apparaissent dans $t$.

# Substitution

## Définition

Une **substitution** $\sigma$ est une fonction d'un ensemble fini de variables ($\subset \mathcal{V}$) dans $\mathcal{T}$.

## Exercice :

Étendre cette définition sur l'ensemble des termes.

**Notation :** Si $t[x_1, ..., x_n]$ est un terme, on note

$$t[\sigma] \text{ ou bien } t[x_1 \leftarrow \theta_1, .., x_n \leftarrow \theta_n]$$

où $\theta_i := \sigma(x_i)$ la substitution obtenue à partir de $t$ en remplaçant chaque occurance de $x_i$ par le terme $\theta_i$.

**Exemple :** On considère le terme $f(x, h(y))$ et la substitution

$$\sigma = [x \leftarrow i(y), y \leftarrow 42]$$

Alors

$$\begin{aligned} \sigma\left(f(x, h(y))\right) &= f\left(\sigma\left(x\right), \sigma\left(h(y)\right)\right) && (1)\\ &= f(\sigma(x), h(\sigma(y)) && (2)\\ &= f(i(y), h(42)) && (3) \end{aligned}$$

**Attention :** Substitutions parallèles, et non pas séquentielles.

$$f(x, h(y))[\sigma] \neq f(i(42), h(42))$$

# Unificateur

**Définition**

Un **unificateur** des termes $t_1, t_2$ est une substitution $\sigma$ telle que

$$\sigma(t_1) = \sigma(t_2)$$

**Problème d'unification :** Soient deux termes $t_1, t_2$.
Un problème d'unification est de la forme $t_1 \stackrel{?}{=} t_2$
Il s'agit de trouver une substitution $\sigma$ telle que $\sigma(t_1) = \sigma(t_2)$.

**Idée :**

- Décomposer les termes tant que les fonctions sont égales.
- *fail* si les fonctions sont différentes.
- Mémoriser la substitution si l'un des termes est une variable.

# Langages de programmation : Filtrage

**Question typique :** Étant donné le code :

```
match Node(3, Leaf 1, Leaf 2) with
    | Leaf x -> x
    | Node(y, nd, Leaf lf) -> lf
```

Quelle valeur est renvoyée ?

C'est un problème d'unification :

- `Leaf x` $\stackrel{?}{=}$ `Node(3, Leaf 1, Leaf 2)` **échec**
- `Node(y, nd, Leaf lf)` $\stackrel{?}{=}$ `Node(3, Leaf 1, Leaf 2)`

Unificateur : $[y \leftarrow 3, nd \leftarrow Leaf\ 1\ , lf \leftarrow 2]$
Valeur renvoyée : 2

# Langages de programmation : Inférence de types

Cf. Cours de Jan-Georg Smaus

**Question typique :** Est-ce que la fonction

```
List.rev : 'a list -> 'a list
```

est applicable à `[2;3] : int list` ?

C'est un problème d'unification :

$$\texttt{'a list} \stackrel{?}{=} \texttt{int list}$$

Réponse : Unificateur $[\texttt{'a} \leftarrow \texttt{int}]$
D'où : `List.rev [2;3] : int list`

**Exemple :**

`Unif (int -> (int -> bool) , (int * int ) -> bool)` ?

`Unif(int, (int * int))` = $fail$

Finalement, on ne peux pas unifier `int -> (int -> bool)` et `(int * int ) -> bool`.

$\rightarrow$ Il n'est pas toujours possible d'unifier deux termes !

**Exercice :**

Est-il possible d'unifier `'a -> bool` et `int -> 'b` ?

# Preuves : Unifier hypothèses et conclusion

**Question typique :** Est-ce que la conclusion est bien une conséquence des hypothèses ?

$$Q(x), P(f(y)) \vdash P(f(a))$$

C'est un problème d'unification :

- $Q(x) \stackrel{?}{=} P(f(a))$ **échec**
- $P(f(x)) \stackrel{?}{=} P(f(a))$ unificateur $[\mathtt{x} \leftarrow \mathtt{a}]$

# Preuves : La méthode de résolution

**Définition**

Un **littéral** est une formule atomique $A$ ou sa négation $\neg A$.
Une **clause** est un ensemble de littéraux.

Une clause $\{A, B\}$ représente la disjonction de ses éléments : $A \vee B$. Un ensemble de clauses représente la conjonction de chacune des clauses.

**Exemple :** $(A \vee B) \wedge (C \vee D) \wedge (\neg C \vee B)$ se réécrit $\{\{A, B\}, \{C, D\}, \{\neg C, B\}\}$.

Résolution des clauses $\{C, D\}$ et $\{\neg C, B\}$ :

$$\frac{\{C, D\}\ \{\neg C, B\}}{\{D, B\}}$$

En général, si $F$ et $G$ sont unifiables avec un unificateur $\sigma$ :

$$\frac{\{F_1, .., F_n, F\}\ \{\neg G, G_1, .., G_n\}}{\{F_1, .., F_n, G_1, .., G_m\}}\ \sigma$$

**Exemple :**

$$\frac{\{P(a)\}\ \{\neg P(x), Q(f(x))\}}{\dfrac{\{Q(f(a)\}\qquad\qquad\qquad\{\neg Q(y)\}}{\{\}}\ \sigma=[y\leftarrow f(a)]}\ \sigma=[x\leftarrow a]$$

La résolution est un mécanisme essentiel du langage *Prolog*.
Cf. Cours Programmation logique, $2^{nd}$ semestre

# Réécriture

**Question typique :** Étant donné la règle de réécriture

$$\texttt{length(x@y)} = \texttt{length(x)} + \texttt{length(y)}$$

Comment montrer que `length(a@b)`= `length(b@a)` ?

Cf. La suite du cours

Panorama des applications :

- Langages de programmation : Inférences de types, filtrage
- Preuves : Unifier Hyp/Conclusion, Méthode de résolution
- Réécriture

# Substitution plus générale

## Définition

La substitution $\sigma$ est **plus générale** que $\sigma_0$ s'il existe $\sigma'$ tel que

$$\sigma_0 = \sigma' \circ \sigma$$

**Exemple :** Soit $\sigma = [x \leftarrow g(y)]$. $\sigma$ est plus générale que $\sigma_0 = [x \leftarrow g(5), z \leftarrow 3]$. Ici, $\sigma' = [y \leftarrow 5, z \leftarrow 3]$.
Voyons sur un exemple,
$\sigma_0 (f(x, z)) = f(g(5), 3)$
et d'autre part,
$\sigma' \circ \sigma (f(x, z)) = \sigma' (f(g(y), z)) = f(g(5), 3)$

# Unificateur le plus général

**Rappel :** Un unificateur des termes $t_1, t_2$ est une substitution $\sigma$ telle que $\sigma(t_1) = \sigma(t_2)$.

## Définition

On appelle **unificateur le plus général** (noté **mgu**) de deux termes $t_1, t_2$, l'unificateur qui est plus général que tout autre unificateur de $t_1$ et de $t_2$.

**Exemple :** On considère le problème d'unification suivant :

$$g(x, f(y)) \stackrel{?}{=} g(x, f(f(z)))$$

Alors le *mgu* est $[y \leftarrow f(z)]$.
Mais d'autres unificateurs, moins généraux, sont par exemple :
$[y \leftarrow f(4); z \leftarrow 4]$, $[y \leftarrow f(z), x \leftarrow 7]$.

# Variables libres

**Définition**

L'ensemble des **variables libres** d'un terme $t$, noté $FV(t)$, est l'ensemble des variables qui apparaissent dans $t$.

**Exemple :**

$FV(f(x, g(y))) = \{x, y\}$ et $FV(f(h(x, x), c)) = \{x\}$

**Exercice :**

Écrire une fonction qui calcul $FV$.

# Algorithme : l'idée

- L'algorithme simplifie des paires $(E, S)$, où :
  · $E$ est un multi-ensemble d'équations à résoudre
  · $S$ est un ensemble de solutions
- Les règles de simplification sont de la forme $(E, S) \Rightarrow (E', S')$.
- Un ensemble de solutions $S$ à la forme $\{x_1 = s_1, .., x_n = s_n\}$ tel que
  · tous les $x_i$ sont différents
  · aucun des $x_i$ n'apparaît dans l'un des $s_j$.

**L'idée :** étant donnés deux termes $t_1, t_2$ à unifier,
- L'algorithme simplifie $(t_1 \stackrel{?}{=} t_2, \{\})$ jusqu'à $(\{\}, \{x_1 = s_1, .., x_n = s_n\})$ ou termine avec un échec.
- En cas de non-échec, le *mgu* est $[x_1 \leftarrow s_1, .., x_n \leftarrow s_n]$.

# Algorithme : Règles de simplification

- `Decompose` : $(\{f(s_1,..,s_n) \stackrel{?}{=} f(t_1,..,t_n)\} \cup E, S) \Rightarrow$
  $(\{s_i \stackrel{?}{=} t_i, \forall i \in \{1,..,n\}\} \cup E, S)$
- `Clash` : $(\{f(s_1,..,s_n) \stackrel{?}{=} g(t_1,..,t_m)\} \cup E, S) \Rightarrow \mathit{fail}$
- `Delete` : $(\{x \stackrel{?}{=} x\} \cup E, S) \Rightarrow (E, S)$
- `Check` : $(\{x \stackrel{?}{=} t\} \cup E, S) \Rightarrow \mathit{fail}$
  si $t \neq x$ et $x \in FV(t)$
- `Eliminate` :
  $(\{x \stackrel{?}{=} t\} \cup E, S) \Rightarrow (E[x \leftarrow t]; S[x \leftarrow t] \cup \{x = t\})$
  si $t \neq x$ et $x \notin FV(t)$

**Remarque :** Algorithme non déterministe
$\rightarrow$ On peut appliquer les règles dans n'importe quel ordre.

**Exemple :**

$(\{p(x,2) \stackrel{?}{=} p(2,x)\};\{\ \})$
$\Rightarrow (\{x \stackrel{?}{=} 2, 2 \stackrel{?}{=} x\};\{\ \})$ (Decompose)
$\Rightarrow (\{2 = 2\};\{x = 2\})$ (Eliminate)
$\Rightarrow (\{\ \};\{x = 2\}$ (Delete)
Donc $\sigma = [x \leftarrow 2]$.

**Exercice :**

Faire l'exemle suivant : $f(x,y) = f(y,g(x))$.

# Terminaison

**Théorème**

L'algorithme d'unification termine.

**Une idée de la preuve :** Après l'application d'une règle,
· Le nombre de variables dans $E$ décroit

OU · Le nombre de variables dans $E$ reste constant, mais la taille des termes décroît

OU · Le nombre de variables et la taille des termes restent égaux, mais le nombre d'équations décroît

Ordre lexico-graphique et multi-ensemble, voir fin du chapitre.

# Correction et Complétude

Soient deux termes $s, t$ :

- Correction : Est-ce que l'algorithme fournit une substitution $\sigma$ telle que $\sigma(s) = \sigma(t)$ ?
- Complétude : Est-ce qu'il fournit un unificateur si $s$ et $t$ sont unifiables ?

**Théorème**

Pour deux termes $s$ et $t$, l'algorithme d'unification est **correct** et **complet**, il calcule le *mgu* de $s$ et $t$.

**Une idée de la preuve :** On raisonne par induction sur la longueur de la dérivation, en utilisant le lemme suivant :

**Lemme**

· Si $(E, S) \Rightarrow (E', S')$, alors l'ensemble des unificateurs de $(E, S)$ est égal à l'ensemble des unificateurs de $(E', S')$.
· Si $(E, S) \Rightarrow \texttt{fail}$, alors $(E, S)$ n'a pas d'unificateurs.

Il faut prouver ce lemme pour chaque règle.

# Limites de l'unification syntaxique

## Unification syntaxique :

On prend en compte uniquement la structure des termes.

**Limites :** On ne peut pas unifier
- $x_f(5) = 5$
- $x \oplus 2 = y \oplus 3$

## $\rightarrow$ Unification d'ordre supérieur :

On prend en compte la sémantique d'exécution des fonctions

**Exemple :** Trouver la fonction $x_f$ telle que $x_f(5) = 5$
- (*Imitation*) La fonction constante 5 : $x_f$ = `fun y -> 5`.
- (*Projection :*) La fonction qui renvoie son argument : $x_f$ = `fun y -> y`.

→ Unification "modulo" :

On prend en compte la "signification" de certains opérateurs.

**Exemple :** Trouver un unificateur $x \oplus 2 = y \oplus 3$ si $\oplus$ est un opérateur commutatif quelconque.

· Solution : $[x \leftarrow 3; y \leftarrow 2]$, parce que $3 \oplus 2 = 2 \oplus 3$.

· Attention $[x \leftarrow 1; y \leftarrow 0]$ n'est pas une solution, parce que $\oplus$ n'est pas $+$ !

Objectif : Faire des preuves équationnelles.

**Exemple :** On représente les entiers naturels ($\mathbb{N} = \{0, 1, 2, ...\}$) en base **"unaire"** avec deux constructeurs `zero` (noté par la suite 0), et `succ` (successeur).
Par exemple, avec ces constructeurs, 1 est représenté par `succ 0` et 2 est représenté par `succ(succ 0)`...
L'addition add peut se définir à l'aide des deux égalités suivantes :

```
add 0 Y = Y et add (succ X) Y = succ (add X Y) (⋆)
```

Comment montrer alors la propriété

```
add (succ 0) (succ(succ 0)) = succ(succ(succ 0)) ?
```

**Remarque :** Un ensemble d'équations comme ($\star$) est appelé une *théorie équationnelle*.

Idée : Orienter les équations

**Exemple :** Pour l'addition,

```
add 0 Y → Y
 add (succ X) Y → succ (add X Y)
```

et appliquer les équations uniquement dans le sens $\rightarrow$.

**Exemple :**

```
add (succ 0) (succ(succ 0)) → succ (add 0 succ(succ
0)) → succ(succ(succ 0)))
```

**Remarque :** Les équations orientées sont appelées des *règles de réécriture*. Un *système de réécriture* est une théorie équationnelle muni de règles de réécriture.

# Exemple (projet)

À l'issue de votre projet, voici un exemple de ce que vous pourrez obtenir :

exemple.rew

```
rules

  // arithmetic functions
  add0: add 0 Y -> Y;
  adds: add (succ X) Y -> succ (add X Y);

trace * ;

simp [left_outer] add 3 4
```

```
Laura@Bob ~/projet01
$ ./eqprover Examples/exemple.rew
Simplifying: ((add 3) 4)
--adds-->
(succ ((add 2) 4))
--adds-->
(succ (succ ((add 1) 4)))
--adds-->
(succ (succ (succ ((add 0) 4))))
--add0-->
7
Result: 7

Laura@Bob ~/projet01
$
```

# Les questions à se poser

- Confluence : Est-ce qu'on peut appliquer les règles dans n'importe quel ordre ?
- Complétude : Est-ce que l'orientation des équations n'entraîne pas une perte d'information ?
- Terminaison : Est-ce que le processus de réécriture s'arrête ?

# Rappel : Termes et substitution

## Définition

L'ensemble $\mathcal{T}$ des **termes** est défini inductivement par :

- un ensemble de variables $\mathcal{V}$
- Si $f$ est un symbole d'arité $k$ et si $t_1, ..t_k \in \mathcal{T}$, alors $f(t_1, .., t_k) \in \mathcal{T}$.

## Définition

Une **substitution** $\sigma$ est une fonction d'un ensemble fini de variables ($\subset \mathcal{V}$) dans $\mathcal{T}$.

**Remarque :** On peut étendre cette définition sur l'ensemble des termes.

# Rappel (suite) :

**Exemple :** On considère le terme $f(x, h(y))$ et la substitution

$$\sigma = [x \leftarrow i(y), y \leftarrow 42]$$

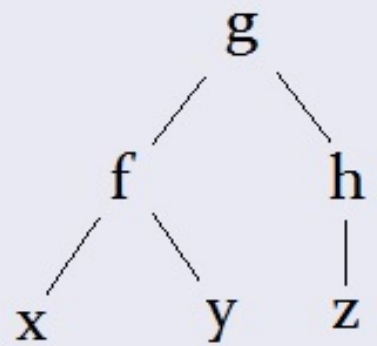
Alors

$$\begin{aligned} \sigma\left(f(x, h(y))\right) &= f\left(\sigma\left(x\right), \sigma\left(h(y)\right)\right) && (4)\\ &= f(\sigma(x), h(\sigma(y)) && (5)\\ &= f(i(y), h(42)) && (6) \end{aligned}$$

# Positions

Chaque terme peut être représenté à l'aide d'un **arbre** :

**Exemple :** $g(f(x,y),h(z))$




## Définition

L'ensemble des **positions** d'un terme $s$ est défini par induction de la façon suivante :

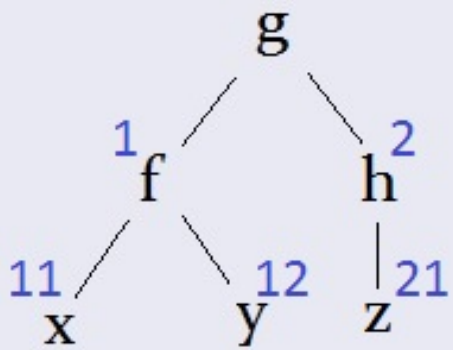
- $s$ est une variable, $Pos(s) = \{\epsilon\}$
- $s = f(t_1, .., t_n)$, $Pos(s) = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{ip/p \in Pos(t_i)\}$.

**Remarque :** La taille de $s$ est $|s| = \#Pos(s)$.

**Notation :** $s|_p$ est le sous-terme de $s$ à la position $p$.

**Exemple (suite) :** On s'interesse toujours au terme

$$s := g(f(x, y), h(z))$$



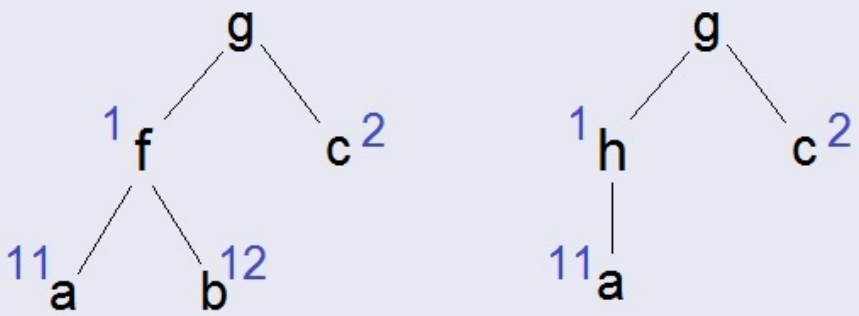

On a alors $Pos(s) = \{\epsilon, 1, 2, 11, 12, 21\}$.
$z$ est à la position [21] donc $g(f(x, y), h(z))|_{21} = z$
$g(f(x, y), h(z))$ est à la position $[\epsilon]$ et $f(x, y)$ est à la position [1].

# Remplacement

[Important]

**Notation :** $s[t]_p$ est le terme obenu en remplaçant $s|_p$ par $t$.

**Exemple (suite) :** $g(f(a,b),c)[h(a)]_1 = g(h(a),c)$




## Exercice :

Effectuer les remplacements suivants :
$f(f(x,x),y)[h(z)]_{12}$ , $f(f(x,x),y)[h(z)]_2$ et $f(f(x,x),y)[h(z)]_1$

# Règle

## Définition

Une **règle** est de la forme $l \to r$ qui vérifie :

- $l$ et $r$ sont des termes
- $l$ n'est pas une variable
  (un système avec une règle $x \to t$ ne terminerait jamais)
- $Var(r) \subset Var(l)$
  (on ne génère pas de variables)

**Exemple :** $f(x) \to g(y)$ n'est pas valide.

# Application d'une règle à la racine

## Données :

- Une règle de réécriture $l \to r$
- Terme $t$ à réécrire

## Méthode :

- Trouver une substitution $\sigma$ telle que $\sigma(l) = t$.
- Le résultat est $\sigma(r)$

## Exercice :

Soit $R = \{f(x, b) \to g(x)\}$.
Lorsque c'est possible, réécrire les termes suivants :

- $f(a, b)$
- $g(b)$
- $f(g(a), b)$
- $f(g(a), c)$
- $f(f(a, a), b)$

# Application d'une règle à la position $p$

## Données :

- Règle de réécriture $l \to r$
- Terme $t$ à réécrire
- Position $p$

## Méthode :

- Trouver une substitution $\sigma$ telle que $\sigma(l) = t|_p$
- Le résultat est $t[\sigma(r)]_p$

## Exercice :

Soit $R = \{g(x) \to h(x, x)\}$.
Réécrire, lorsque c'est possible, les termes suivants :

- $f(g(a), b)$ à la position 1
- $f(g(a), b)$ à la position 2
- $g(g(a))$ à la position 1
- $f(g(a), g(b))$ à la position 1, puis à la position 2

**Attention !**

Lors de la réécriture, la substitution s'applique uniquement à la règle et non pas au terme à réécrire.

**Exercice**

Soit $R = \{g(x, a) \to f(x)\}$. Réécrire les termes suivants :

$$h(g(a, x), x) \text{ et } h(g(b, a), x)$$

# Relation d'équivalence

## Définition

Un ensemble de règles $R = \{l_1 \to r_1, .., l_n \to r_n\}$ engendre les relations suivantes sur les termes :

- $s \to_R t$ si $s \to t$ à l'aide de l'un des $l_i \to r_i \in R$
- $\overset{+}{\to}_R$ est la *fermeture transitive* de $\to_R$
- $\overset{*}{\to}_R$ est la *fermeture réflexive et transitive* de $\to_R$
- $\overset{*}{\leftrightarrow}_R$ est la *fermeture réfléxive, transitive et symétrique* de $\to_R$

On omet l'indice $R$ si l'ensemble des règles est sous-entendu.

## Vrai ou Faux ?

Soit $R = \{f(f(x)) \to f(x), f(a) \to b\}$ un ensemble de règles. Les assertions suivantes sont-elles vraies ou fausses ?

- $f(f(a)) \to f(b)$
- $f(f(a)) \to f(a)$
- $f(f(a)) \xrightarrow{+} f(f(a))$
- $f(f(a)) \xrightarrow{+} b$
- $f(f(a)) \xrightarrow{*} f(f(a))$
- $f(f(a)) \xrightarrow{*} b$
- $f(a) \xrightarrow{*} f(b)$
- $f(b) \xrightarrow{*} f(a)$
- $f(a) \xleftrightarrow{*} f(b)$

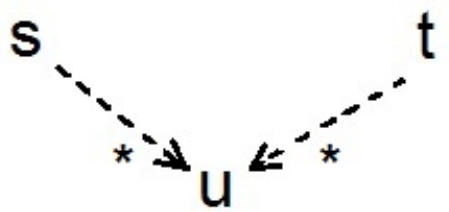
# Formes normales

## Définition

- Un terme $s$ est **réductible** si $\exists t$ tel que $s \rightarrow_E t$
- $s$ est une forme normale s'il n'est pas réductible
- Deux termes $s$ et $t$ sont joignables si $\exists u$ tel que

$$s \overset{*}{\rightarrow}_E u \text{ et } t \overset{*}{\rightarrow}_E u$$

dans ce cas, on écrit $s \downarrow t$.




Joignables

## Exercice :

On considère l'ensemble de règles suivant :
$R = \{f(f(x)) \to f(x), f(a) \to b\}$
Les termes suivants sont-ils réductibles ?

- $f(f(a))$
- $f(b)$
- $f(a)$
- $b$
- $f(x)$
- $f(f(f(b)))$

Les termes suivants sont-ils joignables ?

- $f(a)$ et $f(f(f(a)))$
- $f(f(a))$ et $f(f(x))$
- $f(a)$ et $b$

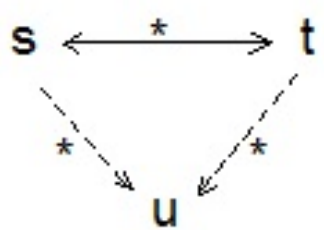
# Church-Rosser et confluence

Soit $\rightarrow_E$ un système de réécriture.

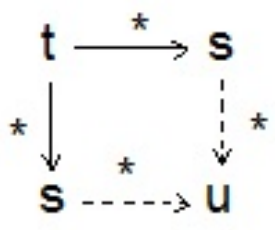
## Définitions

On dit que $\rightarrow_E$ est

- de **Church-Rosser** si $s \overset{*}{\leftrightarrow} t \Rightarrow s \downarrow t$
- **confluente** si $s_1 \overset{*}{\leftarrow}_E t \overset{*}{\rightarrow}_E s_2 \Rightarrow s_1 \downarrow s_2$




Church-Rosser




Confluence

**Pause culture :** Alonzo Church (1903-1995) et John Barkley Rosser (1907-1989) - fondateurs du $\lambda$-calcul

**Théorème**

$\rightarrow_E$ est de Church-Rosser si et seulement si $\rightarrow_E$ est confluent.

On souhaite démontrer ce théorème. Pour cela, on introduit la notion auxiliaire de *semi-confluence* :

$$\rightarrow_E \text{ est } \textbf{semi-confluent} \text{ si } (s \rightarrow t_1 \text{ et } s \overset{*}{\rightarrow} t2 \Rightarrow t1 \downarrow t2.$$

**Exercice :**

1. Montrer que la propriété de Church-Rosser implique la confluence
2. Montrer que la confluence implique la semi-confluence
3. Montrer que la semi-confluence implique la propriété de Church Rosser

**Conséquence :** Dans un système de réduction confluent, un terme a au plus une forme normale.

**Exercice :**

Faire la preuve du corollaire.

# Rappels

Si $R = \{l_1 \to r_1, .., l_n \to r_n\}$ engendre une relation

## Définition

- $\overset{*}{\to}_R$ est la fermeture réflexive et transitive de $\to_R$
- $\overset{*}{\leftrightarrow}_R$ est la *fermeture réfléxive, transitive et symétrique* de $\to_R$

Un terme $s$ est **réductible** si $\exists t$ tel que $s \to_E t$. Sinon, on dit que $s$ est une forme normale.

## Définition

Deux termes $s$ et $t$ sont joignables si $\exists u$ tel que

$$s \overset{*}{\to}_E u \text{ et } t \overset{*}{\to}_E u$$

dans ce cas, on écrit $s \downarrow t$.

# Rappels : Church-Rosser et confluence

Soit $\rightarrow_E$ un système de réécriture.

## Définitions

On dit que $\rightarrow_E$ est

- de **Church-Rosser** si $s \overset{*}{\leftrightarrow} t \Rightarrow s \downarrow t$
- **confluente** si $s_1 \overset{*}{\leftarrow}_E t \overset{*}{\rightarrow}_E s_2 \Rightarrow s_1 \downarrow s_2$

## Théorème

$\rightarrow_E$ est de Church-Rosser si et seulement si $\rightarrow_E$ est confluent.

## Preuve du sens direct $\Rightarrow$ :

Montrer que la propriété de Church-Rosser implique la confluence

## Preuve du sens indirect $\Rightarrow$ :

On introduit la notion auxiliaire de *semi-confluence* :

$$\rightarrow_E \text{ est } \textbf{semi-confluent} \text{ si } (s \rightarrow t_1 \text{ et } s \overset{*}{\rightarrow} t2) \Rightarrow t1 \downarrow t2.$$

1. Montrer que la confluence implique la semi-confluence
2. Montrer que la semi-confluence implique la propriété de Church Rosser

**Conséquence :** Dans un système de réduction confluent, un terme a au plus une forme normale.

## Exercice :

Faire la preuve du corollaire.

# Problèmes indécidables

Les problèmes suivants sont indécidables :

## Confluence

i. Instance : Un système de réécriture $\rightarrow_E$.
Problème : Est-ce que $\rightarrow_E$ est confluent ?

## Terminaison

ii. Instance : Un système de réécriture $\rightarrow_E$.
Problème : Est-ce que $\rightarrow_E$ termine ?

iii. Instance : Un système de réécriture $\rightarrow_E$ et un terme $t$.
Problème : Est-ce que toutes les réductions commençant par $t$ terminent ?

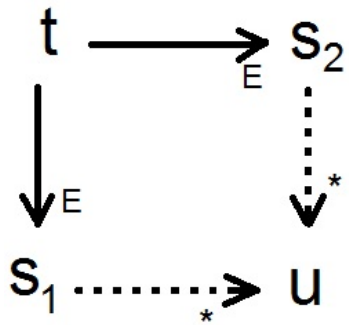
$\rightarrow$ Heureusement, on peut prouver la confluence et la terminaison dans certains cas particuliers.

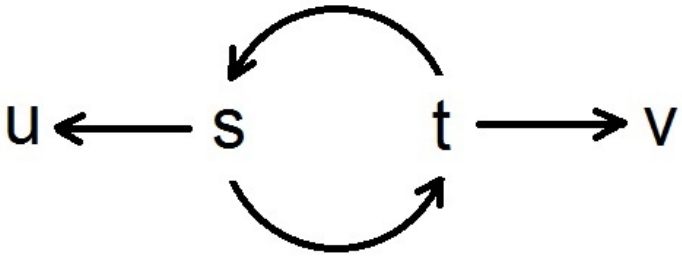
# Confluence locale

## Définition

$\to_E$ est **localement confluent** si

$$s_1 \ {}_E\!\leftarrow t \to_E s_2 \Rightarrow s_1 \downarrow s_2$$




**Attention :** Confluence $\Rightarrow$ Confluence locale
Mais la réciproque est **fausse**

## Lemme (Newman-admis)

Si $\to_E$ termine, alors

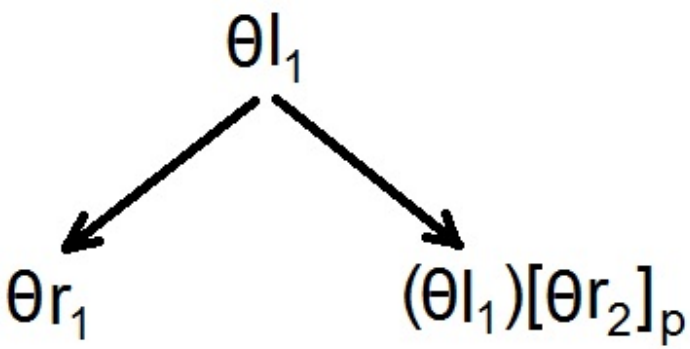
$\to_E$ est confluent $\iff$ $\to_E$ est localement confluent.

# Paires critiques

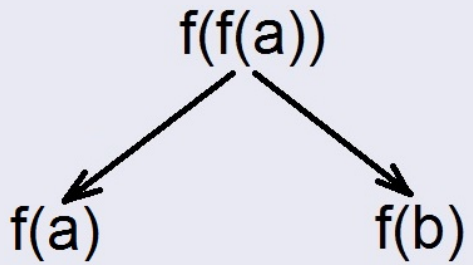
$l_1 \to r_1$ et $l_2 \to r_2$ deux règles de $E$ sans variables communes.

**Définition**

Soit $p \in Pos(l_1)$ tel que $l_1|_p$ n'est pas une variable et soit $\theta$ un unificateur principal de $l_1|_p$ et $l_2$. On dit alors que $(\theta r_1, (\theta l_1)[\theta r_2]_p)$ est une **paire critique**.




**Idée :** Soit
$E = \{f(f(x)) \to f(x), f(a) \to b\}$




$(f(a), f(b))$ est une paire critique.

## Exercice :

Soit $E = \{f(f(x,y),z) \to f(x,f(y,z)), f(g(a),a) \to b\}$.
À l'aide du terme

$$t := f(f(g(a),a),z)$$

Construiser une paire critique.

## Théorème des paires critiques (admis)

Un système de réécriture est localement confluent si et seulement si ses paires critiques sont joignables.

# Où on en est ?

## Bilan :

- Définition de **confluence locale**

**Lemme de Newman :** Si $\rightarrow$ termine,
confluence $\iff$ confluence locale.

- Définition de **paire critique**

**Théorème des paires critiques :**
Confluence locale $\iff$ paires critiques joignables.

# Ordre bien fondé

**Définition**

Un ordre strict $>$ est **bien fondé**, s'il n'existe pas de suite infinie décroissante.

**Exemple :** $(\mathbb{N}, >)$ est un ordre bien fondé. $(\mathbb{Z}, >)$ ne l'est pas.

**Application :** Prouver la terminaison de fonctions.

**Exercice :**

Prouver que la fonction suivante termine :

```
let rec even n =
  if (n=0) then true
  else if (n=1) then false
  else even (n-2)
```

→ À partir d'un ordre bien fondé, comment peut-on en construire d'autres ? (Annexe B)

# Ordres sur le termes

## Définition

Un ordre strict $>$ sur $\mathcal{T}$ est de **réécriture** si
$>$ est *compatible avec les fonctions* :

$$s_1 > s_2 \Rightarrow f(t_1, .., t_{i-1}, s_1, t_{i+1}, .., t_n) > f(t_1, .., t_{i-1}, s_2, t_{i+1}, .., t_n)$$

et $>$ est *clos par substitution* :

$$s_1 > s_2 \Rightarrow \sigma(s_1) > \sigma(s_2)$$

$\rightarrow$ Un ordre strict $>$ sur $\mathcal{T}$ est de **réduction** s'il est de réécriture et bien fondé.

**Exemple :** Soit $t$ un terme, on note $|t|$ la longueur de $t$. On définit un ordre strict sur $\mathcal{T}$ par :

$$s > t \iff |s| > |t|$$

C'est un ordre bien fondé et compatible avec les fonctions.
Cependant, il n'est pas clos par substitution :

$$|f(f(x,x),y)| = 5 > 3 = |f(y,y)|$$

mais pour la substitution $\sigma := \{y \mapsto f(x,x)\}$, on a

$$|\sigma(f(f(x,x),y))| = |f(f(x,x),f(x,x))| = 7$$

$$|\sigma(f(y,y))| = |f(f(x,x),f(x,x))| = 7$$

## Exercice

Pour un terme $t$ et une variable $x$, on note $|t|_x$ le nombre d'occurrences de $x$ dans $t$. On définit un ordre strict $>$ sur $\mathcal{T}$ par :

$$s > t \iff |s| > |t| \text{ et } \forall x \in \mathcal{V}, |s|_x \geq |t|_x$$

Montrer que cet ordre est de réduction.

Finalement,

## Théorème

Un système de réécriture $\rightarrow_E$ est terminant si et seulement si il existe un ordre de réduction $>$ tel que $l > r$ pour tout $l \rightarrow r \in E$.

Entrée : Une théorie équationnelle $E$ et un ordre de réduction $>$ sur les termes $\mathcal{T}$.
Sortie : Un système de réécriture $R$ **convergent** (= confluent et qui termine) équivalent à $E$, *dans le meilleur des cas*.

Les principales idées :

1. Orienter les égalités en utilisant l'ordre de réduction $>$ :
   Soit $\{s = t\} \in E$. Si $s > t$, alors $s \to t$. Si $s < t$, $t \to s$.
   *Problème : Il se peut qu'on ne puisse pas orienter les égalités.*
2. Analyser les paires critiques ($\to$ confluence locale)
   Une fonction auxiliaire $CP(R)$ calcule les paires critiques de $R$, non joignables.
   *Complétion :* Ajout de nouvelles règles pour rendre les paires critiques joignables.

**Initialisation :**

Si toutes les $\{s = t\} \in E$ sont orientables
alors $R_0 :=$ orienter $E$
sinon échec ;

**Faire :**

$R_{i+1} := R_i$
pour tout $(s, t) \in CP(R_i)$
réduire $s, t$ sous formes normales $\hat{s}, \hat{t}$
si on peut orienter $(\hat{s}, \hat{t})$ en $l \to r$
alors $R_{i+1} := R_{i+1} \cup \{l \to r\}$
sinon échec
$i := i + 1$
**tant que** $R_i \neq R_{i-1}$ ;

Renvoyer $R_i$

**Résultats possibles de l'algorithme :**

- Un ensemble $R$ convergent (= confluent et qui termine) équivalent à $E$, c'est à dire $s \overset{*}{\leftrightarrow} t \iff s =_E t$.
- échec, si on ne peut pas orienter certaines égalités. ( $\rightarrow$ l'ordre de réduction $>$ est peut-être mal choisi)
- Non-terminaison de l'algorithme...

**Exercice :**

Soit $E = \{f(f(x)) = f(x), g(x) = f(f(x)), g(g(x)) = x\}$. On considère $>$ l'ordre de réduction défini plus tôt par :
$s > t \iff |s| > |t|$ et $\forall x \in \mathcal{V}, |s|_x \geq |t|_x$
Déterminer un système de réécriture $R$ équivalent à $E$.

# Correction

**Initialisation :** Orienter les égalités de $E$

$$R_0 := \{f(f(x)) \to f(x), f(f(x)) \to g(x), g(g(x)) \to x\}$$

**Étape 1 :**

Paire(s) critique(s) non joignable(s) : $(f(x), g(x))$
Mise sous forme normale : -rien à faire-
Orientation : $f(x) \to g(x)$ ( !)

$$R_1 := \{f(f(x)) \to f(x), f(f(x)) \to g(x), g(g(x)) \to x, f(x) \to g(x)\}$$

## Étape 2 :

Paire(s) critique(s) non joignable(s) :
$(f(x), f(g(x))), (g(x), f(g(x)))$
Mise sous forme normale : $(g(x), x), (g(x), x)$
Orientation : $g(x) \to x$

$$R_2 := \{f(f(x)) \to f(x), f(f(x)) \to g(x), g(g(x)) \to x,$$
$$f(x) \to g(x), g(x) \to x\}$$

## Étape 3 :

Paire(s) critique(s) non joignable(s) : pas d'autres paires critiques
$\to$ Fin de l'algorithme $R = R_2$.

## Analyse du résultat :

$$R := \{f(f(x)) \to f(x), f(f(x)) \to g(x), g(g(x)) \to x,$$
$$f(x) \to g(x), g(x) \to x\}$$

Il y a des règles surperflues :

- $g(g(x)) \to x$ résulte de $g(x) \to x$
- $f(f(x)) \to f(x)$ résulte de $f(x) \to g(x)$ puis $g(x) \to x$
- $f(f(x)) \to g(x)$ résulte de $f(x) \to g(x)$ $(\times 2)$, $g(x) \to x$

Finalement,

$$R = \{f(x) \to g(x), g(x) \to x\}$$

Objectif de la réécriture : faire des preuves équationnelles

Soit $E$ une théorie équationnelle, on veut montrer que $s =_E t$.

Procédure de Knuth-Bendix : $R$ convergent et équivalent à $E$.

Il faut montrer alors $s \overset{*}{\leftrightarrow}_R t$.

$\rightarrow_R$ à la propriété de Church-Rosser (par théorème de CR)

1. Réduire $s \overset{*}{\rightarrow} s'$, avec $s'$ irréductible (la réduction termine !)
2. De même, réduire $t \overset{*}{\rightarrow} t'$, avec $t'$ irréductible
3. Si $s' = t'$ alors $s \overset{*}{\leftrightarrow} t$. Sinon (non $s \overset{*}{\leftrightarrow} t$).

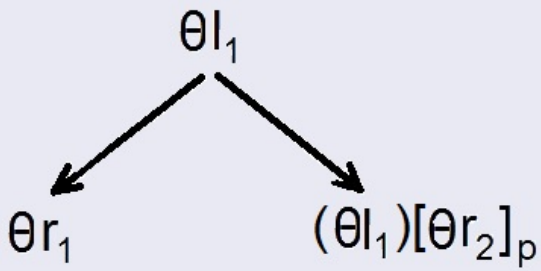
**Exemple (suite) :** On considère toujours
$E = \{f(f(x)) = f(x), f(f(x)) = g(x), g(g(x)) = x\}$ et
$R = \{f(x) \to g(x), g(x) \to x\}$.
Prouvons $f(f(x)) = g(f(x))$

1. Par raisonnement équationnel :
   $f(f(x)) = f(f(f(x))) = g(f(x))$
   *Nécessite la découverte d'un terme intermédiaire plus complexe*
2. Par réduction : $f(f(x)) \overset{*}{\to} x, g(f(x)) \overset{*}{\to} x$
   *Procédure automatisée*

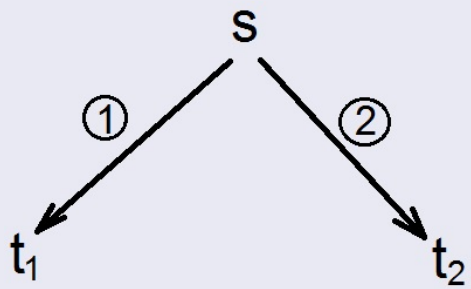
**Exercice :** Peut-on prouver $f(a) = g(f(b))$ ?

# Preuve du théorème des paires critiques

**Preuve ($\Rightarrow$) :** Si le système est localement confluent, par définition, toute paire critique est joignable car issue de dérivations de la forme :




**Preuve ($\Leftarrow$) :** Supposons maintenant que les paires critiques sont joignables. Soit $s$ un terme tel que :




avec $l_1 \xrightarrow{1} r_1$ et $l_2 \xrightarrow{2} r_2$.

On sait que $\forall i \in \{1,2\}, \exists p_i \in Pos(s)$ telle que

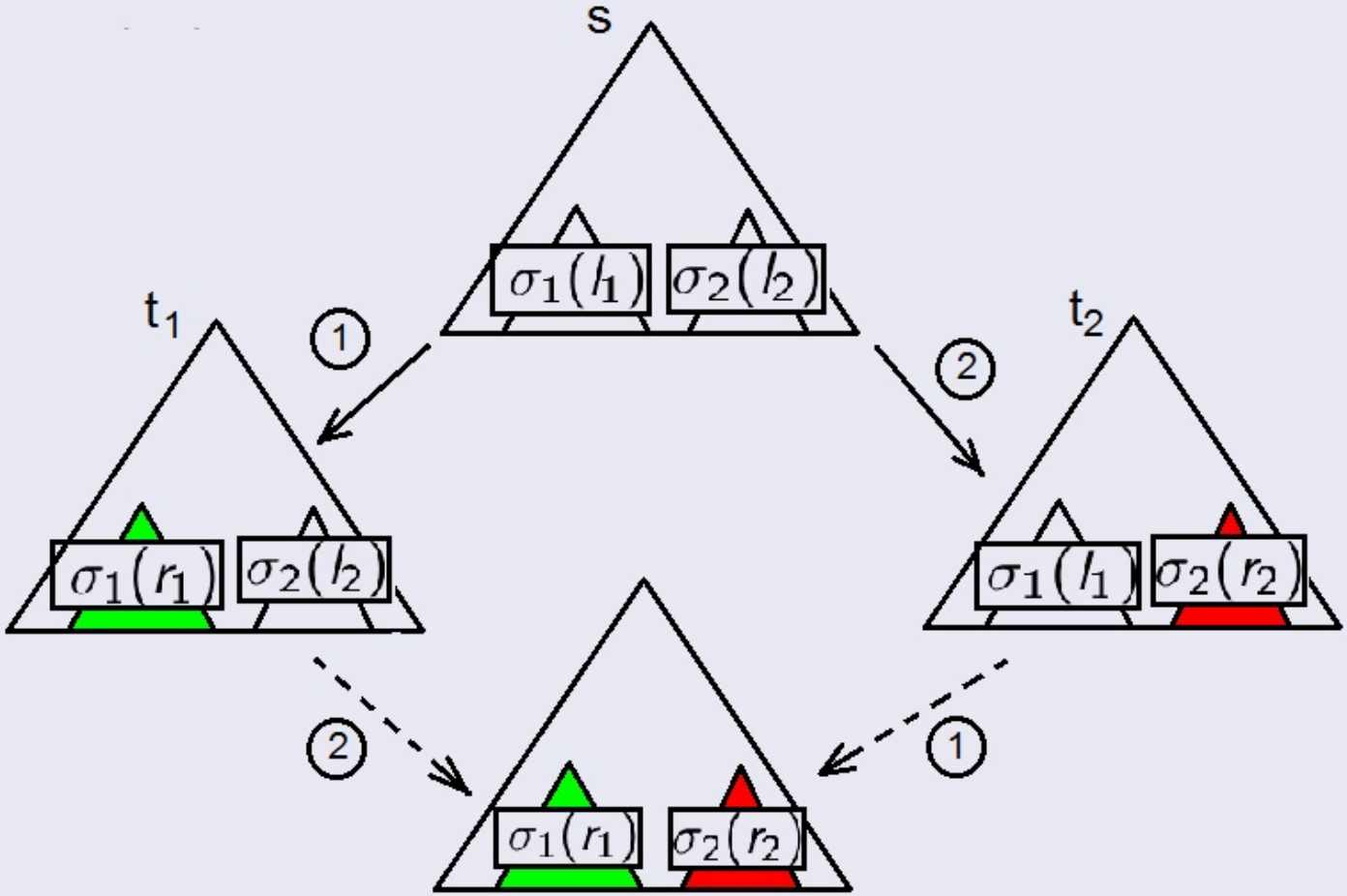
$$s|_{p_i} = \sigma_i(l_i) \text{ et } t_i = s[\sigma_i(r_i)]_{p_i}.$$

$\rightarrow$ Il faut montrer que $t_1$ et $t_2$ sont joignables.

La suite de la démonstration dépend des positions $p_1$ et $p_2$.

**Preuve ($\Leftarrow$, suite) :**

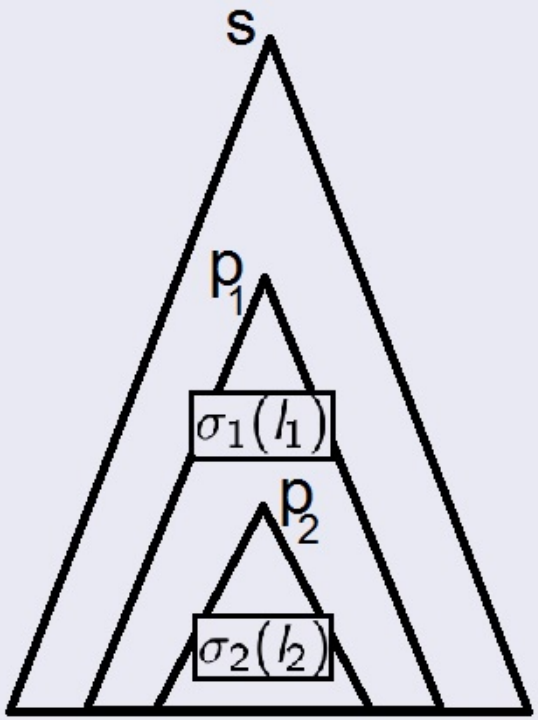
Cas 1 : $p_1$ et $p_2$ sont des sous-arbres de $s$ séparés.




$\rightarrow$ $t_1$ et $t_2$ sont joignables. Localement confluent.

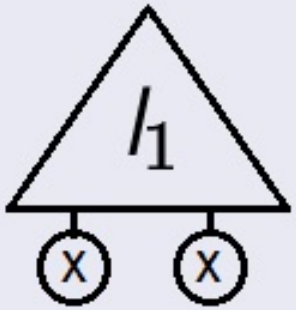
**Preuve ($\Leftarrow$, suite) :**

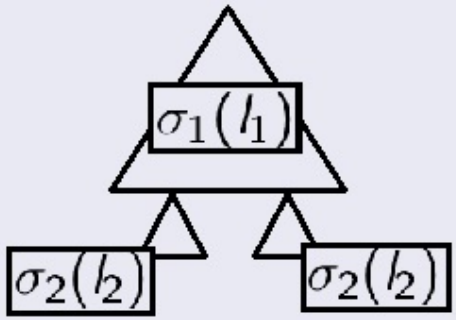
Cas 2 : Si $p_1$ est un préfixe de $p_2$ (sous-arbres qui se chevauchent)




a) Chevauchement non critique

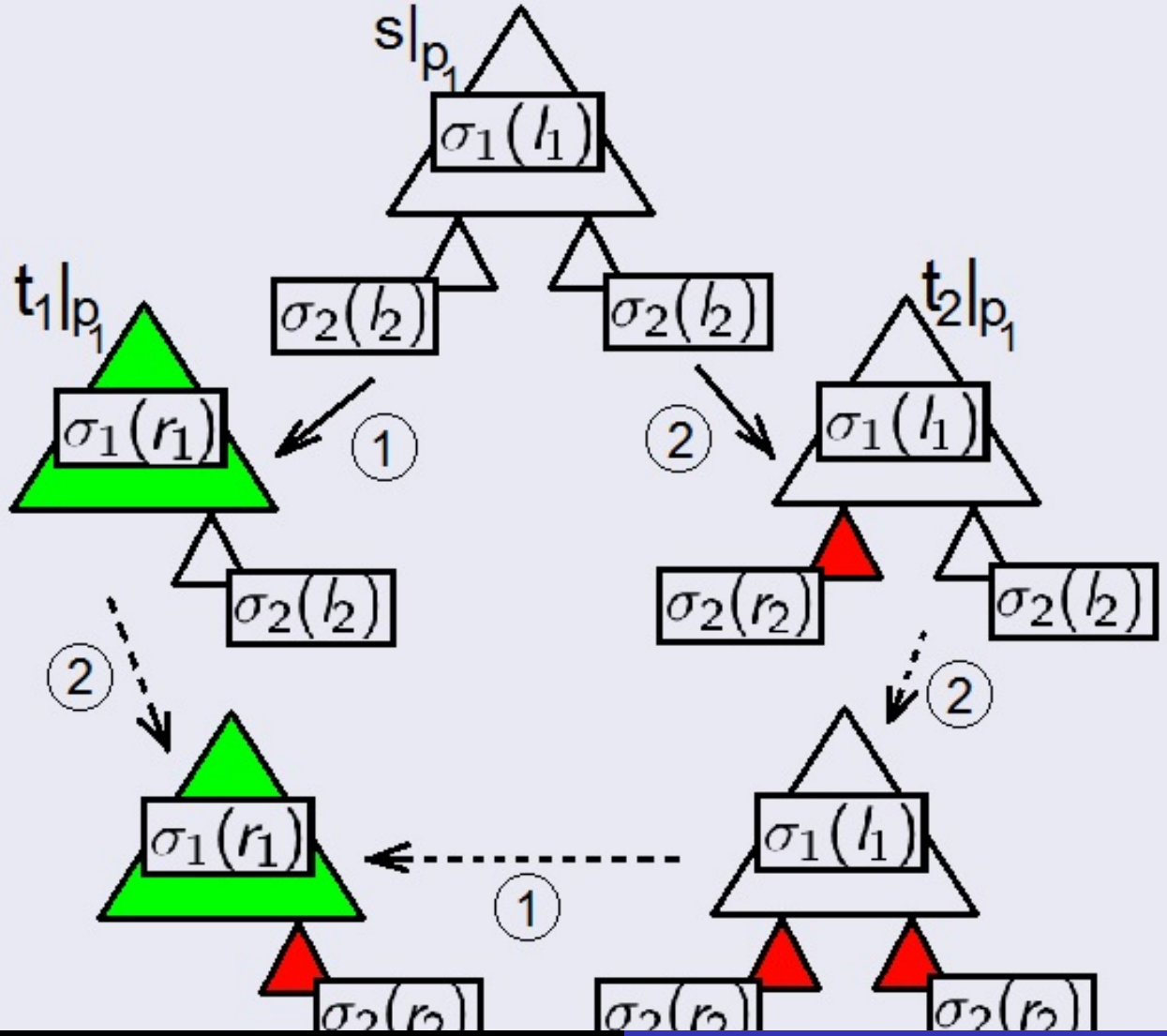
$\sigma_2(l_2)$ ne chevauche pas $l_1$ lui-même mais est contenu dans $\sigma_1$



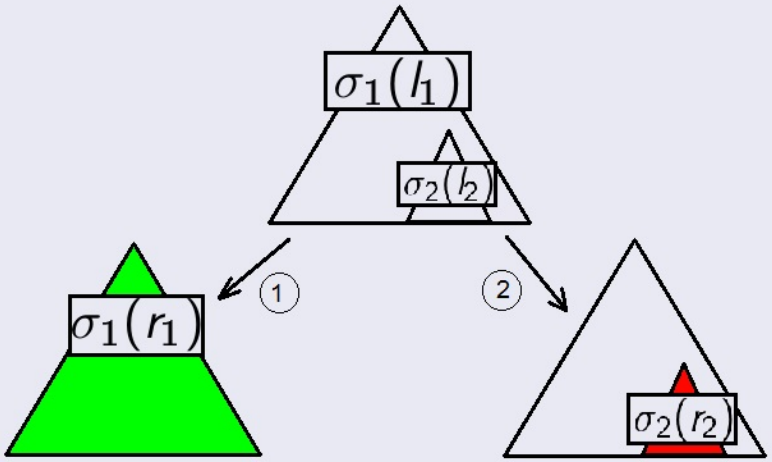




$x$ représente une variable du support de $\sigma_1$, c'est une feuille de $l_1$.

## Preuve ($\Leftarrow$, suite) : Cas 2, a) Chevauchement non critique



**Preuve ($\Leftarrow$, suite) :** Cas 2,
b) Chevauchement critique




La position de $\sigma_2(l_2)$ dans $l_1$ n'est pas une variable : $\sigma_2(l_2)$ chevauche $l_1$.

$\rightarrow (\sigma_1(r_1), (\sigma_1(l_1))[\sigma_2(r_2)])$ est une paire critique.
Et par hypothèse, toutes les paires critiques sont joignables.

Dans les trois cas, $t_1$ et $t_2$ sont joignables.
Finalement, le système est bien localement confluent.

# Méthode 1 : Image inverse

Soient $(A, >_A)$ et $(B, >_B)$, deux ensembles munis d'une relation d'ordre. Soit $f : A \to B$ une fonction strictement croissante, c'est à dire

$$\forall a_1, a_2 \in A ;\ a_1 >_A a_2 \Rightarrow f(a_1) >_B f(a_2)$$

**Proposition**

Si $>_B$ est un ordre bien fondé sur $B$, alors $>_A$ est bien fondé sur $A$.

## Exercice :

Considérons la fonction even sur les nombres entiers relatifs ($\mathbb{Z}$) :

```
let rec even n =
  if (n=0) then true
  else if ((n=1) or (n=-1)) then false
  else if (n<0) then even (n+2) else even (n-2)
```

En appliquant le principe de l'image inverse, prouver la terminaison de cette fonction.

# Méthode 2 : Ordre lexicographique

## Définition

Soient $(A, >_A)$ et $(B, >_B)$ deux ensembles munis d'un ordre.
**L'ordre lexicographique** $>_{lex}$ est défini sur $A \times B$ par :

$$(a_1, b_1) >_{lex} (a_2, b_2) \iff (a_1 >_A a_2) \text{ ou } ((a_1 = a_2) \text{ et } (b_1 >_B b_2))$$

## Lemme

Si $>_A$ et $>_B$ sont bien fondés, alors $>_{lex}$ est un ordre bien fondé.

## Exercice :

Prouver le lemme

## Exemple

La **fonction d'Ackermann** est définie par :

```
let rec ack = function
    (0,y) -> y + 1
  | (x,0) -> ack(x-1, 1)
  | (x,y) -> ack(x-1 , ack(x,y-1))
```

1928 - W. Ackermann

# Méthode 3 : Multiensembles

Soient $(A, >_A)$ et $\mathcal{M}$ l'ensemble des multiensembles sur $A$.

## Définition

On définit un ordre $>_{mult}$ sur $\mathcal{M}$ par :
$M >_{mult} N$ s'il existe deux multiensembles $X, Y$, tels que

- $N = (M \setminus X) \cup Y$
- $X \neq \emptyset$
- $\forall y \in Y, \exists x \in X, x >_A y$

## Lemme

Si $>_A$ est un ordre bien fondé, alors $>_{mult}$ est également bien fondé.

**Exemples :**

- $\{5,3,2\} >_{mult} \{5,2,2\}$
- Est-ce que $\{5,3,2\} >_{mult} \{4,4,4,2,2\}$ ?
- Est-ce que $\{5,3,2\} >_{mult} \{5,5\}$ ?

**Exercice :**

Refaire la preuve de terminaison de l'algorithme d'unification.