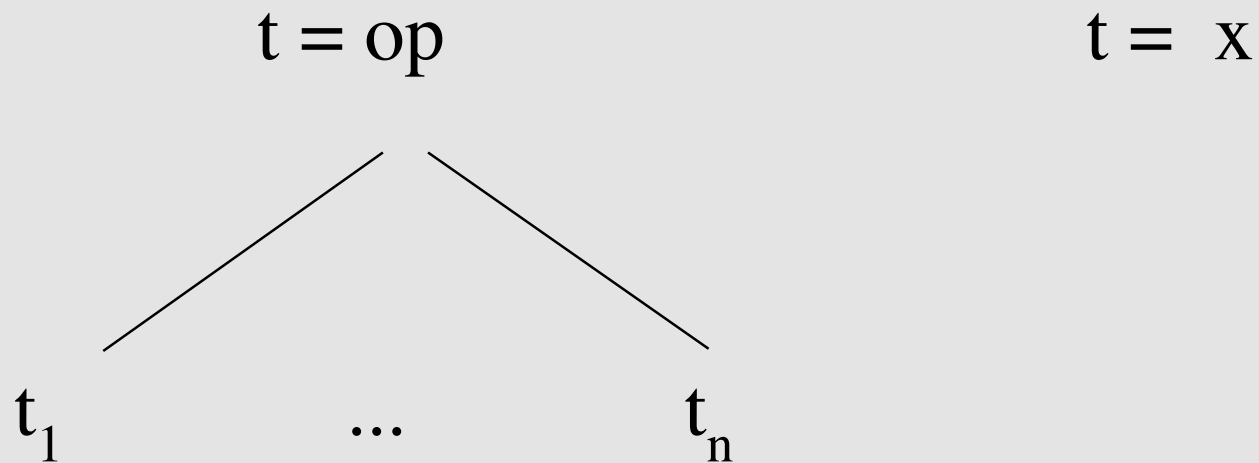


Filtrage

Implantation en Caml

Termes formels

terme := <variable>
<opérateur> <liste de termes>



Filtrage

Implantation en Caml

Termes formels

terme := <variable>
 <opérateur> <liste de termes>

Le type terme est paramétré par:

'a, le type des opérateurs

'b, le type des noms de variables

Filtrage

Implantation en Caml

Termes formels

terme := <variable>
 <opérateur> <liste de termes>

Le type terme est paramétré par:

'a, le type des opérateurs

'b, le type des noms de variables

```
#type ('a,'b) term = Term of 'a * ('a,'b) term list  
                 | Var of 'b
```

Filtrage

Implantation en Caml

Termes *vs* arbres

#type ('a,'b) term =

Term of 'a * ('a,'b) term list | Var of 'b;;

Cf les arbres quelconques

type 'a tree = Tr of 'a * 'a tree list;;

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$$t = \text{op}(t_1, \dots, t_n)$$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{lr}(t) = (\text{lr } t_1) (\text{lr } t_2) \dots (\text{lr } t_n)$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{lr}(t) = (\text{g e } (\text{lr } t_1)) (\text{lr } t_2) \dots (\text{lr } t_n)$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{lr}(t) = (\text{g } (\text{g e } (\text{lr } t_1)) (\text{lr } t_2)) \dots (\text{lr } t_n)$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{lr}(t) = (\text{g } (\text{g } (\text{g } e \text{ (lr } t_1)) \text{ (lr } t_2)) \dots \text{ (lr } t_n))$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{lr}(t) = \text{f op } (\text{g } (\text{g } (\text{g e } (\text{lr } t_1)) (\text{lr } t_2)) \dots (\text{lr } t_n))$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{lr}(t) = \text{f op} (\text{g} (\text{g} (\text{g e} (\text{lr } t_1)) (\text{lr } t_2)) \dots (\text{lr } t_n))$

$t = x \longrightarrow t = \text{Var } x$

Filtrage

Implantation en Caml

Parcours de termes (gauche – droite)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{lr}(t) = \text{f op } (g (g (g e (\text{lr } t_1)) (\text{lr } t_2)) \dots (\text{lr } t_n))$

$t = x \longrightarrow t = \text{Var } x$

$\text{lr}(t) = v x$

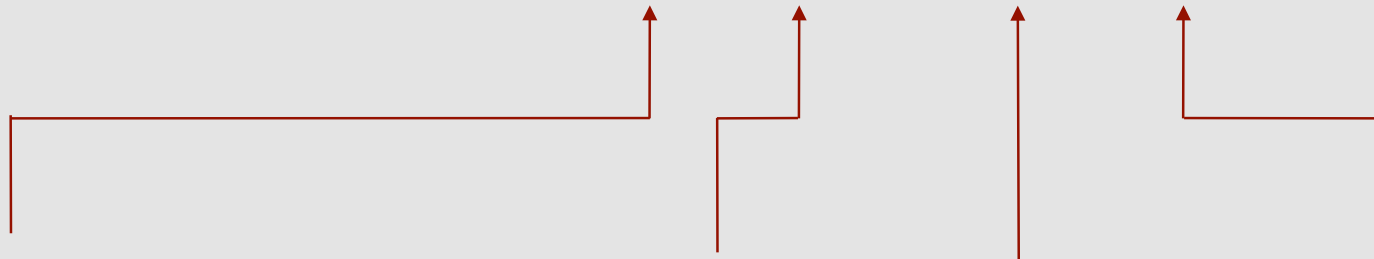
Filtrage

Implantation en Caml

Parcours de termes (gauche - droite)

```
#type ('a,'b) term = Term of 'a*('a,'b) term list | Var of 'b;;
```

lr = lr_term f g e v



Gère l'opérateur et le
résultat du parcours des
ss-termes immédiats

Fct itérée sur
les ss-termes
immédiats

Elt initial
de l'itération

traite les
variables



Filtrage

Implantation en Caml

Parcours de termes (gauche - droite)

lr = lr_term f g e v

Gère l'opérateur et le
résultat du parcours des
ss-termes immédiats

Fct itérée sur
les ss-termes
immédiats

Elt initial
de l'itération

traite les
variables

Term(op, l)

Var(x)

f op (List.fold_left (fun x t ->g x (lr t)) e l)

(v x)

Filtrage

Implantation en Caml

Parcours de termes (gauche - droite)

```
#type ('a,'b) term = Term of 'a*('a,'b) term list | Var of 'b;;
```

```
#let lr_term f g e v =  
  let rec lr t = match t with  
    Term(op,l) -> f op (List.fold_left (fun x t ->g x (lr t)) e l)  
  | Var x      -> v x  
  in lr;;
```

```
val lr_term : ('a -> 'b -> 'c) ->  
('b -> 'c -> 'b) -> 'b -> ('d -> 'c) -> ('a, 'd) term -> 'c = <fun>
```

Filtrage

Implantation en Caml

Parcours de termes (droite – gauche)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{rl}(t) = (\text{rl } t_1) (\text{rl } t_2) \dots (\text{rl } t_n)$

Filtrage

Implantation en Caml

Parcours de termes (droite – gauche)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{rl}(t) = (\text{rl } t_1) (\text{rl } t_2) \dots (\text{g } (\text{rl } t_n) \text{ e})$

Filtrage

Implantation en Caml

Parcours de termes (droite – gauche)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{rl}(t) = (\text{rl } t_1) (\text{g } (\text{rl } t_2) \dots (\text{g } (\text{rl } t_n) \text{e}) \dots)$

Filtrage

Implantation en Caml

Parcours de termes (droite – gauche)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{rl}(t) = (\text{g} (\text{rl } t_1) (\text{g} (\text{rl } t_2) \dots (\text{g} (\text{rl } t_n) \text{e} \dots)))$

Filtrage

Implantation en Caml

Parcours de termes (droite – gauche)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{rl}(t) = \mathbf{f\ op} \ (\mathbf{g\ (rl\ t_1)} \ (\mathbf{g\ (rl\ t_2)} \ \dots \ (\mathbf{g\ (rl\ t_n)} \ \mathbf{e}) \ \dots))$

Filtrage

Implantation en Caml

Parcours de termes (droite – gauche)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{rl}(t) = \text{f op (g.rl } t_1 \text{ (g.rl } t_2 \text{ ... (g.rl } t_n \text{ e) ...))}$

Filtrage

Implantation en Caml

Parcours de termes (droite – gauche)

$t = \text{op}(t_1, \dots, t_n) \longrightarrow t = \text{Term}(\text{op}, [t_1; \dots; t_n])$

$\text{rl}(t) = \mathbf{f\ op} (\text{g}\cdot\text{rl } t_1 (\text{g}\cdot\text{rl } t_2 \dots (\text{g}\cdot\text{rl } t_n \text{ e}) \dots))$

$t = x \longrightarrow t = \text{Var } x$

$\text{rl}(t) = \mathbf{v\ x}$

Filtrage

Implantation en Caml

Parcours de termes (droite - gauche)

```
#type ('a,'b) term = Term of 'a*('a,'b) term list | Var of 'b;;  
#let (%) f g x = f (g x);;
```

```
let rl_term f g e v =  
  let rec rl t = match t with  
    Term(op, l) -> f op (List.fold_right (g % rl) l e)  
  | Var x      -> v x  
  in rl;;
```

```
val rl_term : ('a -> 'b -> 'c) ->  
('c -> 'b -> 'b) -> 'b -> ('d -> 'c) -> ('a, 'd) term -> 'c = <fun>
```

Filtrage

Implantation en Caml

Ensemble des variables d'un terme

$\text{vars}(\text{Term}(\text{op}, [t_1; \dots; t_n])) = \text{vars}(t_1) \cup \dots \cup \text{vars}(t_n).$

$\text{vars}(\text{Var } x) = [x]$

Donc: $f \text{ op } l = 1, g = \text{union}, e = [], v \ x = [x]$

```
#let union l1 l2 =
```

```
List.fold_right (fun a l -> if List.mem a l2 then l else a::l) l1 l2
```

Filtrage

Implantation en Caml

Ensemble des variables d'un terme

$\text{vars}(\text{Term}(\text{op}, [t_1; \dots; t_n])) = \text{vars}(t_1) \cup \dots \cup \text{vars}(t_n).$

$\text{vars}(\text{Var } x) = [x]$

Donc: $f \text{ op } l = 1, g = \text{union}, e = [], \forall x = [x]$

```
let vars = lr_term (fun op l -> l) union [] (fun x -> [x]);;  
val vars : ('a, 'b list) term -> 'b list = <fun>
```

```
#let union l1 l2 =
```

```
List.fold_right (fun a l -> if List.mem a l2 then l else a::l) l1 l2;;
```

Filtrage

Implantation en Caml

Ensemble des variables d'un terme

$$t = x + y * z$$

```
#let t=Term ("+", [Var "x"; Term ("*", [Var "y"; Var "z"])]);;
```

```
val t : (string, string) term = Term ("+", ...
```

```
#vars t;;
```

```
- : string list = ["x"; "y"; "z"]
```

$$t' = x + y * x$$

```
# let t'= Term ("+" ,[Var "x"; Term ("*", [Var "y"; Var "x"])]);;
```

```
val t' : (string, string) term = Term ("+", ...
```

```
# vars t';;
```

```
- : string list = ["y"; "x"]
```

Filtrage

Implantation en Caml

Ensemble des variables d'un terme

$t'' = x + y * c$ où c est une constante

```
# let t'' = Term ("+", [Var "x"; Term ("*", [Var "y"; Term( "c", [ ])])]);;  
val t'' : (string, string) term = Term ("+", ...
```

```
# vars t'';
```

```
- : string list = ["x"; "y"]
```

Filtrage

Implantation en Caml

Occurrence d'une variable dans un terme

```
# let occurs v t = List.mem v (vars t);;  
val occurs : string -> (string, string) term -> bool = <fun>
```

```
# occurs "x" t;;  
- : bool = true
```

```
# occurs "w" t;;  
- : bool = false
```

Filtrage

Implantation en Caml

Substitutions

But Dans un terme, substituer des termes à des variables

Exemple

Substituer $(2+3)$ à x et 4 à y dans $t = x+y*z$

Substitution: termes \rightarrow termes

Définie par sa valeur sur les variables

$$s(x) = 2+3, s(y) = 4, s(z) = z$$

$$s(t) = s(x + y * z) = s(x) + s(y) * s(z) = (2+3) + 4 * z$$

Filtrage

Implantation en Caml

Substitutions

Représentation en Caml

```
#let s = [ ("x",Term ("+", [ Term("2",[ ]) ; Term("3",[ ]) ] ));  
          ("y", Term ("4",[ ]) ) ];;
```

List.assoc

```
# List.assoc 3 [(4,"a"); (2, "b"); (3, "c"); (1, "d")];;  
- : string = "c"
```

```
# List.assoc 8 [(4,"a"); (2, "b"); (3, "c"); (1, "d")];;  
Exception: Not_found.
```

Filtrage

Implantation en Caml

Substitutions

Calcul de l'application de la substitution s à t

Si $t = op(t_1, \dots, t_n)$

$subst(t) = op(subst(t_1), \dots, subst(t_n))$

Si $t = x$

$subst(t) =$ **si** $(x, u) \in s$ **alors** u
 sinon x

Filtrage

Implantation en Caml

Substitutions

Calcul de l'application de la substitution s à t

$\text{Si } t = \text{op}(t_1, \dots, t_n) \quad \text{subst}(t) = \text{op}(\text{subst}(t_1), \dots, \text{subst}(t_n))$
 $\text{Si } t = x \quad \text{subst}(t) = \text{si } (x, u) \in s \text{ alors } u$
 $\quad \quad \quad \text{sinon } x$

En Caml

$\text{subst}(\text{Term}(\text{op}, [t_1; \dots; t_n])) = \text{Term}(\text{op}, [(\text{subst } t_1); \dots; (\text{subst } t_n)])$

$\text{subst}(\text{Var } x) = \text{try}(\text{List.assoc } x \ s) \ \text{with } _ \rightarrow \text{Var } x$

Filtrage

Implantation en Caml

Substitutions en Caml

```
let apply_subst s = rl_term
  (fun op l -> Term(op, l))
  l: liste des sous-termes immédiats après substitution
  (fun t l -> t :: l)
  t est un nouveau terme, l la liste des termes déjà traités
  [ ]
  la liste des termes déjà traités est initialisée à vide
  (fun x -> try List.assoc x s with _ -> Var x) ;;
  l'association de la variable et du terme qui la remplace
```

val apply_subst :

<code>('a * ('b, 'a) term) list -></code>	Substitution
<code>('b, 'a) term -> ('b, 'a) term = <fun></code>	terme -> terme

Filtrage

Implantation en Caml

Substitutions en Caml

```
let apply_subst s = rl_term
  (fun op l -> Term(op, l))
  (fun t l -> t::l)
  []
  (fun x -> try List.assoc x s with _ -> Var x) ;;
```

```
val apply_subst :
('a * ('b, 'a) term) list -> -> ('b, 'a) term -> ('b, 'a) term = <fun>
```

Filtrage

Implantation en Caml

Substitutions en Caml

```
t = x + y * z
```

```
s = [ (x, 2+3); (y, 4) ];;
```

```
# apply_subst s t ;;
```

```
- : (string, string) term =
```

```
Term ("+",
```

```
    [Term ("+", [Term ("2", [ ]); Term ("3", [ ])]);
```

```
      Term ("*", [Term ("4", [ ]); Var "z"])
```

```
    ]
```

```
)
```

```
(2 +3) + (4 * z)
```

Filtrage

Implantation en Caml

Composée de substitutions

s_1 et s_2 : substitutions sous forme de listes

Éléments des listes : couples (x, u) , x variable, u terme

Calcul de $s = s_1 \circ s_2$

s est constituée de :

tous les couples $(x, (s1\ u))$ tels que $(x, u) \in s_2$

Filtrage

Implantation en Caml

Composée de substitutions

s_1 et s_2 : substitutions sous forme de listes

Éléments des listes : couples (x, u) , x variable, u terme

Calcul de $s = s_1 \circ s_2$

s est constituée de :

tous les couples $(x, (s1\ u))$ tels que $(x, u) \in s_2$

$ll = \text{List.map (fun (x,u) -> (x, apply_subst s1 u)) } s2$

Filtrage

Implantation en Caml

Composée de substitutions

s_1 et s_2 : substitutions sous forme de listes

Éléments des listes : couples (x, u) , x variable, u terme

Calcul de $s = s_1 \circ s_2$

s est constituée de :

tous les couples $(x, (s1\ u))$ tels que $(x, u) \in s_2$

$ll = \text{List.map } (\text{fun } (x,u) \rightarrow (x, \text{apply_subst } s1\ u))\ s2$

tous les couples (x, u) de $s1 - s2$ (x invariant par $s2$)

Filtrage

Implantation en Caml

Composée de substitutions

s_1 et s_2 : substitutions sous forme de listes

Éléments des listes : couples (x, u) , x variable, u terme

Calcul de $s = s_1 \circ s_2$

s est constituée de :

tous les couples $(x, (s1\ u))$ tels que $(x, u) \in s_2$

$l1 = \text{List.map } (\text{fun } (x,u) \rightarrow (x, \text{apply_subst } s1\ u))\ s2$

tous les couples (x, u) de $s1 - s2$ (x invariant par $s2$)

$\text{List.find_all } (\text{fun } (x,t) \rightarrow \text{not } (\text{List.mem } x\ \text{var_s2}))\ s1$

```
List.find_all;;
```

```
- : ('a -> bool) -> 'a list -> 'a list = <fun>
```

Filtrage

Implantation en Caml

Composée de substitutions

s_1 et s_2 : substitutions sous forme de listes

Éléments des listes : couples (x, u) , x variable, u terme

Calcul de $s = s_1 \circ s_2$

s est constituée de :

tous les couples $(x, (s1\ u))$ tels que $(x, u) \in s_2$

$l1 = \text{List.map } (\text{fun } (x,u) \rightarrow (x, \text{apply_subst } s1\ u))\ s2$

tous les couples (x, u) de $s1 - s2$ (x invariant par $s2$)

$\text{List.find_all } (\text{fun } (x,t) \rightarrow \text{not } (\text{List.mem } x\ \text{var_s2}))\ s1$

Filtrage

Implantation en Caml

Composée de substitutions

s_1 et s_2 : substitutions sous forme de listes

Éléments des listes : couples (x, u) , x variable, u terme

Calcul de $s = s_1 \circ s_2$

s est constituée de :

tous les couples $(x, (s1\ u))$ tels que $(x, u) \in s_2$

$l1 = \text{List.map } (\text{fun } (x,u) \rightarrow (x, \text{apply_subst } s1\ u))\ s2$

tous les couples (x, u) de $s1 - s2$ (x invariant par $s2$)

$l2 = \text{let var_s2} = \text{List.map fst } s2 \text{ in}$

$\text{List.find_all } (\text{fun } (x,t) \rightarrow \text{not } (\text{List.mem } x\ \text{var_s2}))\ s1$

Filtrage

Implantation en Caml

Composée de substitutions

```
# let compsubst s1 s2 =  
  let l1 = List.map (fun (x,u) -> (x,apply_subst s1 u)) s2  
  and l2 = let var_s2 = List.map fst s2  
           in  
           List.find_all (fun (x,t)-> not (List.mem x var_s2)) s1  
  in l1@l2;;  
  
val compsubst :  
  ('a * ('b, 'a) term) list -> ('a * ('b, 'a) term) list ->  
  ('a * ('b, 'a) term) list = <fun>
```

Filtrage

Implantation en Caml

Filtrage

t : un terme

p : un motif

But : trouver s t.q. $s(p) = t$

- si $p = x$ (*variable*) $-->$ $s = [(x, t)]$
- si $p = \text{op} (p_1, \dots, p_n)$, le filtrage est possible si
 - $t = \text{op} (t_1, \dots, t_n)$
 - $\forall i \in \{1, \dots, n\}$ filtrage possible de t_i par p_i $-->$ s_i
 - Compatibilité des s_i
Si $(x, u) \in s_i$ et $(x, u') \in s_j$, alors $u = u'$

Filtrage

Implantation en Caml

Filtrage

*Rajout d'un couple (x, u) à une substitution s
sous réserve de compatibilité*

```
# exception Match_exc;;  
exception Match_exc
```

```
# let add_subst s (x,u) =  
  try let u' = (List.assoc x s) in  
    if u = u' then s else raise Match_exc  
  with Not_found -> (x,u) :: s;;
```

```
val add_subst : ('a * 'b) list -> 'a * 'b -> ('a * 'b) list = <fun>
```

Filtrage

Implantation en Caml

Filtrage

La fonction combine

```
# List.combine;;  
- : 'a list -> 'b list -> ('a * 'b) list = <fun>  
  
# List.combine [1; 2; 3; 4; 5; 6] ["a"; "b"; "c"; "d"; "e"; "f"];;  
- : (int * string) list =  
      [(1, "a"); (2, "b"); (3, "c"); (4, "d"); (5, "e"); (6, "f")]  
  
# List.combine [1; 2; 3; 4; 5; 6] ["a"; "b"; "c"];;  
Exception: Invalid_argument "List.combine".
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

`match_gen s (p, t)`

- calcule la substitution s' résultat du filtrage du terme t sur le motif p
- en vérifiant à chaque étape la compatibilité avec une substitution s donnée
- renvoie `s@s'`

`('a * ('b, 'c) term) list ->`

`('b, 'a) term * ('b, 'c) term ->`

`('a * ('b, 'c) term) list`

substitution

(motif, terme)

substitution

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

- $p = \text{Var } x \quad \text{-->} \quad \text{add_subst } s (x, t)$
- $p = \text{Term}(op_p, sb_patterns)$ et $t = \text{term}(op_t, sb_terms)$

si $op_p \neq op_t$ alors erreur

sinon

combine sur $sb_patterns = [p_1; \dots; p_n]$ et $sb_terms = [t_1; \dots; t_n]$

si arité différente --> erreur

sinon on itère récursivement sur la liste obtenue:

$[(p_1, t_1) ; \dots ; (p_n, t_n)]$

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =
```

```
substitution motif terme
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =
```

substitution motif terme

filtre le terme t sur le motif p en prenant en compte la substitution s

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with  
  (Var(x), t)
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with  
  (Var(x), t) -> add_subst s (x,t)
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with  
  | (Var(x), t) -> add_subst s (x,t)  
  | (Term (op_p, sb_patterns), Term (op_t, sb_terms)) when op_p=op_t ->
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with  
  | (Var(x), t) -> add_subst s (x,t)  
  | (Term (op_p, sb_patterns), Term (op_t, sb_terms)) when op_p=op_t ->  
    (List.combine sb_patterns sb_terms)
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with  
  | (Var(x), t) -> add_subst s (x,t)  
  | (Term (op_p, sb_patterns), Term (op_t, sb_terms)) when op_p=op_t ->  
    s (List.combine sb_patterns sb_terms)
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with  
  | (Var(x), t) -> add_subst s (x,t)  
  | (Term (op_p, sb_patterns), Term (op_t, sb_terms)) when op_p=op_t ->  
    List.fold_left match_gen  
      s (List.combine sb_patterns sb_terms)
```

Filtrage

Implantation en Caml

Filtrage

match_gen : fonction de filtrage généralisée

```
let rec match_gen s (p, t) =  
  match (p, t) with  
  | (Var(x), t) -> add_subst s (x,t)  
  | (Term (op_p, sb_patterns), Term (op_t, sb_terms)) when op_p=op_t ->  
    List.fold_left match_gen  
      s (List.combine sb_patterns sb_terms)  
  | _ -> raise Match_exc;;
```

Filtrage

Implantation en Caml

Filtrage

matching : fonction de filtrage

```
let rec matching = match_gen [ ];;
```

val matching :

$('a, 'b) \text{ term} * ('a, 'c) \text{ term} \rightarrow$ (motif, terme)
 $('b * ('a, 'c) \text{ term}) \text{ list}$ substitution
 $= \langle \text{fun} \rangle$

Filtrage

Implantation en Caml

Filtrage

matching : fonction de filtrage

```
let rec matching = match_gen [ ];;
```

si $p = x+y*z$ et $t = (a+b)+(x*y)*(a+b)$ on obtient:

```
#matching (p ,t);;
```

```
- : (string * (string, string) term) list =
```

```
["z", Term ("+", [Var "a"; Var "b"]);
```

```
"y", Term ("*", [Var "x"; Var "y"]);
```

```
"x", Term ("+", [+Var "a"; Var "b"])]
```

