(1/2) DM (9/3)

Correction et Complétude de l'algorithme de Martelli-Montanari

Si L'algo. appliqué à S renvoit une substitution σ, alors σ est un unificateur le plus général de S. Inversement, si l'algo renvoit $\perp$, alors $\mathcal{U}(S) = \emptyset$. (Term Rewriting p73→77)

Idée générale de la preuve :

L'algo peut être vu comme la réécriture de S :

$$S = S_0 \to S_1 \to \dots \to \underbrace{S_n = T}_{\text{Une forme normale}}$$

On doit montrer la conservation des unificateurs (2)

(1) On peut extraire directement un m.g.u

Def : On dit que S est en forme résolue si $S = \{x_1 =? t_1, \dots, x_n =? t_n\}$ où les (x_i) sont des variables, 2 à 2 distinctes, n'apparaissant dans aucun des t_i. Dans ce cas on définit la substitution $\vec{S}$:
$\vec{S} = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$

Montrons successivement :

Lemme 1. Si S est résolvable et est en forme normale, alors S est en forme résolue.

On raisonne sur les équations que peut contenir S :

- $x =? x \longrightarrow$ NON car éliminés dans l'algo
- $f(\dots) = g(\dots) \longrightarrow$ idem
- $x =? t$ où $x \in Var(t) \longrightarrow$ NON car dans ce cas S n'est pas résolvable

En effet : $t = f(t_1, \dots t_n)$ avec $x \in Var(t_i)$

Alors $|\sigma(t)| > |\sigma(t_i)| \geq |\sigma(x_i)| \Rightarrow \sigma(t) \neq \sigma(x_i)$ $\forall \sigma$.

2/2

D1

915

Alors S ne contient que des équations de type $x =^? t$ où $x \notin \text{Var}(t)$, CQFD

Pour la suite de la preuve on utilise un lemme intermédiaire

Lemme 2 : Si S est en forme résolue, alors $\forall \sigma \in \mathcal{U}(S)$, $\sigma \vec{S} = \sigma$

Montrons que $\forall x \in V$, $(\sigma \vec{S})(x) = \sigma(x)$. → 2 cas.

Premier cas : $x = x_i$ ⟵ les variables dans S

Alors $S(x_i) = t_i$

$\Rightarrow \sigma \vec{S}(x_i) = \sigma(t_i) = \sigma(x_i)$ ok ($\sigma \in \mathcal{U}(S)$)

Second cas : $x \notin \{x_i\}$: alors $S(x) = x$, le résultat est immédiat.

Lemme 3 : Si S est en forme résolue, alors $\vec{S}$ est un mgu de S

Lemme 2 + Lemme 3 $\Rightarrow$ l'implication ⊂ est montrée.

Preuve du lemme 3 : prendre $\rho = \sigma$. en utilisant le lemme 2.

Montrons à présent la conservation des unificateurs :

Supposons $S \to T$. Il y a 5 cas (les règles dans l'algo)

- $\{t =^? t\} \uplus S \to S$: clair
- $\{f(t_1,\dots,t_n) =^? f(u_1,\dots,u_n)\} \uplus S \to \{t_1 =^? u_1, \dots\} \uplus S$: clair
- $\{x =^? t\} \uplus S \to \bot$ ($x \in \text{Vau}(t)$) : on a vu que $\mathcal{U}(\{x =^? t\} \uplus S) = \emptyset$ → ok.
- $\{f(\vec{t_n}) = g(\vec{u_p})\} \uplus S$: idem.
- $\{x =^? t\} \uplus S \to \{x \mapsto t\} \cup S[t/x]$ ($x \notin \text{Vau}(t)$) : le cas intéressant.

$\sigma \in \mathcal{U}(\{x =^? t\} \uplus S) \iff \sigma x = \sigma t$ et $\sigma \in \mathcal{U}(S)$

$\iff \sigma x = \sigma t$ et $\sigma(t \mapsto x) \in \mathcal{U}(S)$

↳ Lemme 2 appliqué à $S = \{x \mapsto t\}$ résolue

$\iff \sigma x = \sigma t$ et $\sigma \in \mathcal{U}((t \mapsto x) \uplus S)$

$\iff \sigma \in \mathcal{U}(\{x \mapsto t\} \cup S[t/x])$ □

Conclusion : $S \to^* \bot \Rightarrow \mathcal{U}(S) = \emptyset$ (lemme 3) : complétude

(1/3)

D2

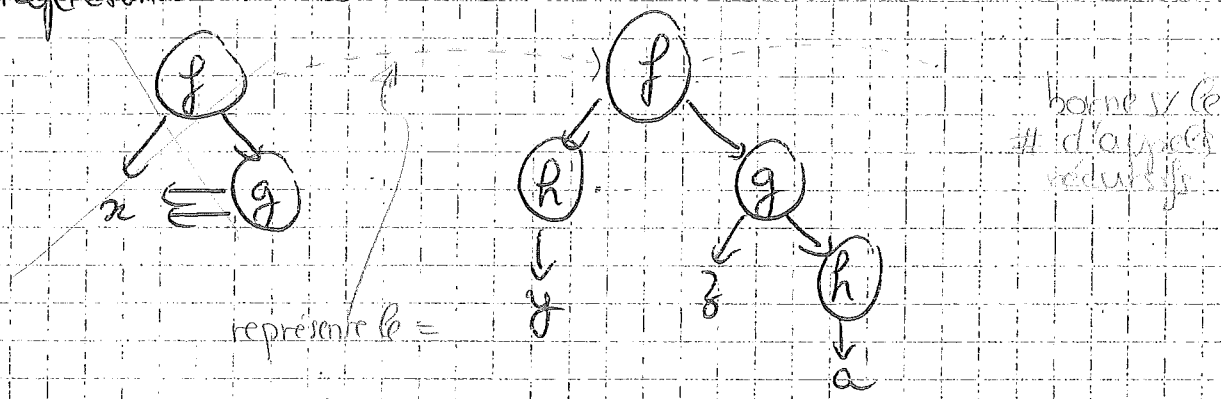
(9/9)

Implémentation de l'algorithme de Martelli Montanari en utilisant une structure de graphe pour représenter le problème d'unification, avec une complexité quadratique

(Term Rewriting, p 87 → 88)

On illustrera notamment cette implémentation avec l'exemple :

$S = \{ f(x, g(x,x)) \stackrel{?}{=} f(h(y), g(z, h(a))) \}$ dont la représentation est :

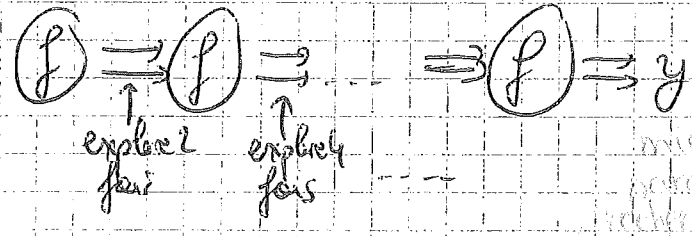


borne sup le # d'appels récursifs

Idée générale • Structure utilisée pour comprimer l'information.

• Mais il existe un autre comportement expérimental : quand on veut vérifier si $x \in Var(t)$, on explore les sous-termes. Si ceux-ci sont égaux, on va effectuer cette étape trop de fois !

ex



Solution : lorsqu'on parcourt un nœud dans une recherche d'occurrences, on le marque avec un `hher`, qui augmentera à chaque nouvelle recherche. Si on parcourt un nœud qui a un marqueur égal au `hher` actuel, inutile de l'explorer de nouveau.

→ la recherche d'occurrences se fait en $O(m)$, où m est le nombre d' du graphe.

2/3

D2

S12

• Lors de l'unification, on voudrait éviter d'explorer plusieurs fois les mêmes sous-termes.

→ Structure d'union-find : on unifie les termes que l'on sait égaux syntaxiquement à un moment donné, et on ne garde qu'un seul représentant.

La fonction d'unification s'écrit alors :

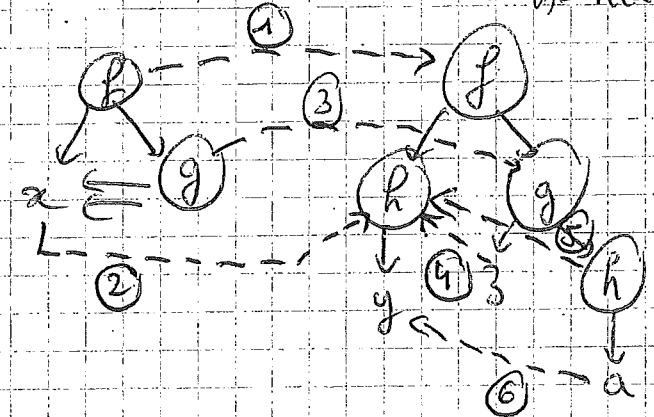
```
Unifier(t,u) =
    t = find(t)   u = find(u)        // On se ramène aux représentants
    • si t=u, retourner vrai
    • si t et u sont des variables
        union(t,u); retourner vrai
    • si t est une variable mais pas u
        si occurs(t,u) (union(t,u); retourner vrai)
    • si u est une variable mais pas t : cas similaire
    • si t et u ne sont pas des variables   // t=f(...) // u=g(...)
        s'ils commencent par le même symbole
            union(t,u); unifierListe(t,u)
        sinon retourner faux
```

Exemple

$f(x, g(x,z)) \stackrel{?}{=} f(h(y), g(3, h(a)))$ ① $\rightarrow \begin{cases} x = h(y) \\ g(x,z) = g(3, h(a)) \end{cases}$

② $\rightarrow g(h(y), h(y)) = g(3, h(a))$ ③ $\rightarrow \begin{cases} h(y) = 3 \\ h(y) = h(a) \end{cases}$ ④ $\rightarrow h(y) = h(a)$

⑤ $\rightarrow y = a$ ⑥ $\rightarrow \emptyset$



On observe alors que lorsqu'on appelle union(t1, t2), alors t2 n'est plus jamais renvoyé par find, et donc n'est plus parcouru par unify.

→ nombre d'appels récursifs à unify : $O(m)$

coût d'1 appel : au maximum

2 find + accès + union

↓ $O(m)$ // Suivre les pointeurs → pas de cycle

↓ $O(m)$

↓ $O(1)$ // crée 1 pointeur

$\underbrace{O(m)}_{O(m)}$ pour une opération

⇒ Algo quadratique. CQFD.

Remarque : Avec une implémentation plus rapide de Union et Find, la complexité d'une séquence de m unions et n find est $O(m\,G(n))$; d'où 1 algo quasi-linéaire sur le même principe

$G(n)$ ↓ ≤ 5 en pratique