Sous-typage uniforme et non-structurel

Stéphane Glondu

Stage réalisé à l'Université de Californie, Davis
sous la direction de Zhendong Su

juin–août 2005

Plan

Introduction au sous-typage

Arbres et types

Un type peut être vu comme un arbre.

Arbres et types

Un type peut être vu comme un arbre.

Pour pouvoir en parler, on a besoin :

Arbres et types

Un type peut être vu comme un arbre.

Pour pouvoir en parler, on a besoin :

- d'une *signature* Σ : un ensemble de symboles

Arbres et types

Un type peut être vu comme un arbre.

Pour pouvoir en parler, on a besoin :

- d'une *signature* Σ : un ensemble de symboles
- pour chaque $f \in \Sigma$, d'une *arité* $\text{arity}(f)$

Arbres et types

Un type peut être vu comme un arbre.

Pour pouvoir en parler, on a besoin :

- d'une *signature* Σ : un ensemble de symboles
- pour chaque $f \in \Sigma$, d'une *arité* $\mathrm{arity}(f)$
- pour chaque $f \in \Sigma$, de *polarités* $\mathrm{pol}(f, i) \in \{-1, 1\}$ pour $1 \leqslant i \leqslant \mathrm{arity}(f)$

Arbres et types

Un arbre est une fonction partielle

$$\tau : (\mathbb{N} - \{0\})^* \to \Sigma$$

qui vérifient quelques propriétés :

Arbres et types

Un arbre est une fonction partielle

$$\tau : (\mathbb{N} - \{0\})^* \rightarrow \Sigma$$

qui vérifient quelques propriétés :

- $\tau(\epsilon)$ est toujours défini

Arbres et types

Un arbre est une fonction partielle

$$\tau : (\mathbb{N} - \{0\})^* \to \Sigma$$

qui vérifient quelques propriétés :

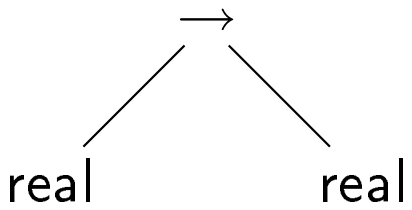
- $\tau(\epsilon)$ est toujours défini
- $\tau(\pi i)$ est défini ssi $\tau(\pi) = f$, et que $1 \leqslant i \leqslant \text{arity}(f)$

Arbres et types

Par exemple, avec $\Sigma = \left\{ \top_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1} \right\}$:

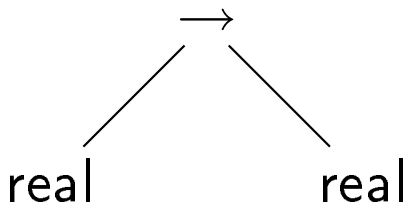
Arbres et types

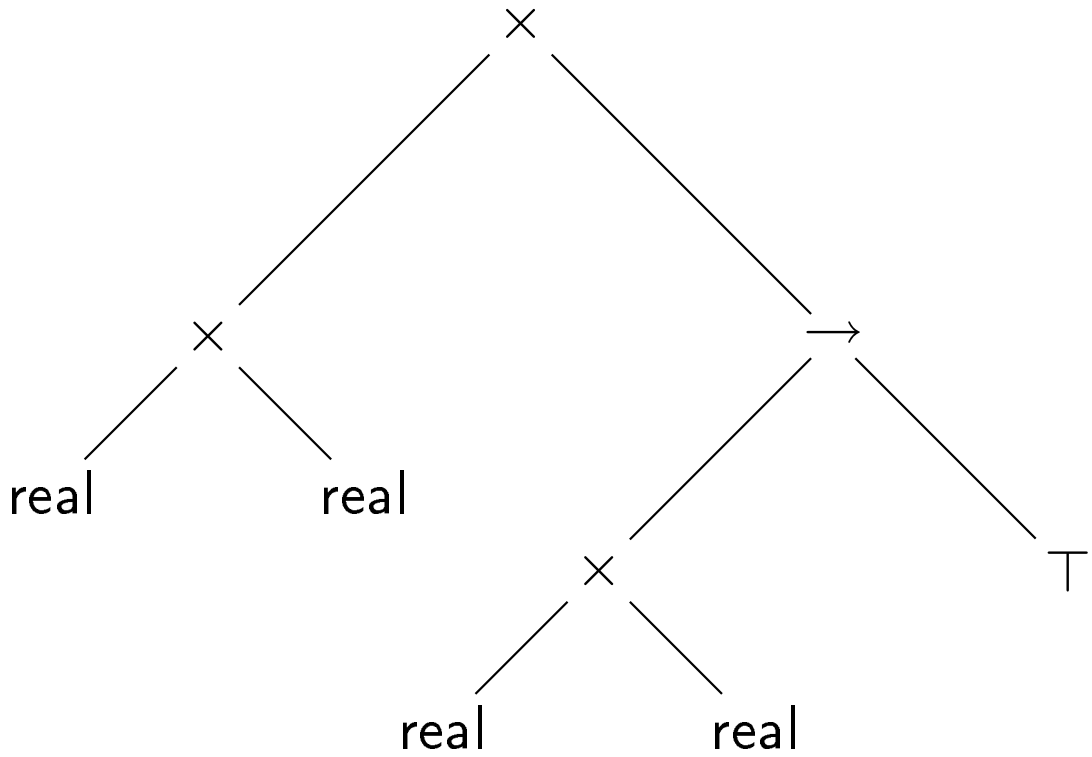
Par exemple, avec $\Sigma = \left\{ \top_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1} \right\}$:



Arbres et types

Par exemple, avec $\Sigma = \left\{ \top_0, \mathrm{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1} \right\}$:





Qu'est-ce que le sous-typage ?

Il s'agit de mettre un ordre partiel sur les types pour modéliser des relations de sous-ensemble entre des types de données.

Sous-typage structurel

Deux types comparables ont la même structure (par exemple, avec $\Sigma = \left\{\mathsf{int}_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1}\right\}$) :

Sous-typage structurel

Deux types comparables ont la même structure (par exemple, avec $\Sigma = \left\{\mathsf{int}_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1}\right\}$) :

- on met un ordre sur les constantes :

Sous-typage structurel

Deux types comparables ont la même structure (par exemple, avec $\Sigma = \left\{\mathsf{int}_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1}\right\}$) :

- on met un ordre sur les constantes : int $\leqslant$ real

Sous-typage structurel

Deux types comparables ont la même structure (par exemple, avec $\Sigma = \left\{\mathsf{int}_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1}\right\}$) :

- on met un ordre sur les constantes : $\mathsf{int} \leqslant \mathsf{real}$
- l'ordre est relevé à tous les termes en respectant les constructeurs et les polarités :

Sous-typage structurel

Deux types comparables ont la même structure (par exemple, avec $\Sigma = \left\{\mathsf{int}_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1}\right\}$) :

- on met un ordre sur les constantes : $\mathsf{int} \leqslant \mathsf{real}$
- l'ordre est relevé à tous les termes en respectant les constructeurs et les polarités :

$$\tau_1 \times \tau_2 \leqslant \tau_1' \times \tau_2' \quad \text{ssi} \quad \tau_1 \leqslant \tau_1' \text{ et } \tau_2 \leqslant \tau_2'$$

Sous-typage structurel

Deux types comparables ont la même structure (par exemple, avec $\Sigma = \left\{\mathsf{int}_0, \mathsf{real}_0, \times_2^{1,1}, \rightarrow_2^{-1,1}\right\}$) :

- on met un ordre sur les constantes : int $\leqslant$ real
- l'ordre est relevé à tous les termes en respectant les constructeurs et les polarités :

$$\tau_1 \times \tau_2 \leqslant \tau_1' \times \tau_2' \quad \text{ssi} \quad \tau_1 \leqslant \tau_1' \text{ et } \tau_2 \leqslant \tau_2'$$

$$\tau_1 \rightarrow \tau_2 \leqslant \tau_1' \rightarrow \tau_2' \quad \text{ssi} \quad \tau_1' \leqslant \tau_1 \text{ et } \tau_2 \leqslant \tau_2'$$

Sous-typage structurel

Des exemples un peu plus concrets :

Sous-typage structurel

Des exemples un peu plus concrets :

- $\text{real} \longrightarrow \text{real} \leqslant \text{int} \longrightarrow \text{real}$

Sous-typage structurel

Des exemples un peu plus concrets :

- $\text{real} \to \text{real} \leqslant \text{int} \to \text{real}$
- $\text{int} \times \text{int} \leqslant \text{real} \times \text{real}$

Sous-typage non-structurel

Comme le sous-typage structurel, mais. . .

Sous-typage non-structurel

Comme le sous-typage structurel, mais. . .on rajoute deux constantes spéciales $\bot$ et $\top$ et la règle :

$$\bot \leqslant \tau \leqslant \top$$

pour tout τ.

Sous-typage non-structurel

Comme le sous-typage structurel, mais. . .on rajoute deux constantes spéciales $\bot$ et $\top$ et la règle :

$$\bot \leqslant \tau \leqslant \top$$

pour tout τ.
Conséquence : deux types comparables n'ont plus la même forme !

Sous-typage non-structurel

Comme le sous-typage structurel, mais. . .on rajoute deux constantes spéciales $\bot$ et $\top$ et la règle :

$$\bot \leqslant \tau \leqslant \top$$

pour tout τ.
Conséquence : deux types comparables n'ont plus la même forme !
Par exemple :

Sous-typage non-structurel

Comme le sous-typage structurel, mais. . .on rajoute deux constantes spéciales $\bot$ et $\top$ et la règle :

$$\bot \leqslant \tau \leqslant \top$$

pour tout τ.
Conséquence : deux types comparables n'ont plus la même forme !
Par exemple :

- lancement d'une exception, récursion mal fondée : $\bot$

Sous-typage non-structurel

Comme le sous-typage structurel, mais. . .on rajoute deux constantes spéciales $\bot$ et $\top$ et la règle :

$$\bot \leqslant \tau \leqslant \top$$

pour tout τ.
Conséquence : deux types comparables n'ont plus la même forme !
Par exemple :

- lancement d'une exception, récursion mal fondée : $\bot$
- () en Caml : $\top$

Sous-typage non-structurel

Comme le sous-typage structurel, mais. . .on rajoute deux constantes spéciales $\bot$ et $\top$ et la règle :

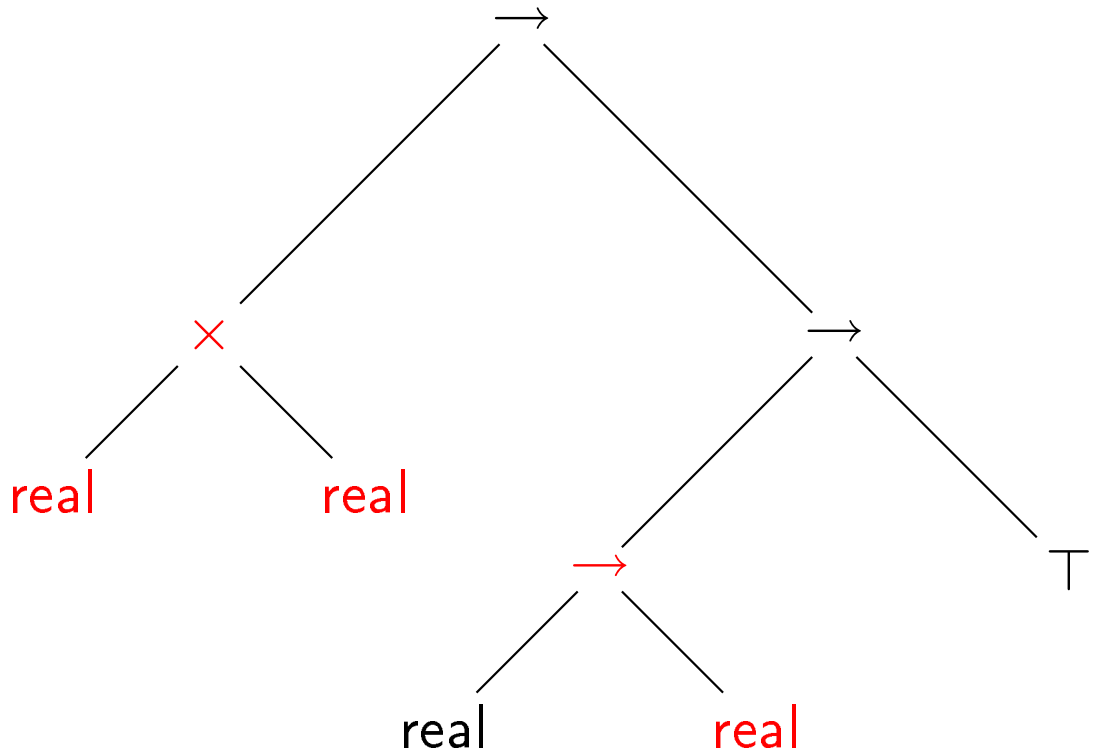
$$\bot \leqslant \tau \leqslant \top$$

pour tout τ.
Conséquence : deux types comparables n'ont plus la même forme !
Par exemple :

- lancement d'une exception, récursion mal fondée : $\bot$
- () en Caml : $\top$
- objets : $\text{int} \times \text{int} \times (\text{int} \times \text{int} \to \top) \leqslant \text{int} \times \text{int} \times \top$

Polarités

Chaque nœud a une *polarité* :



Sous-typage uniforme

Outil permettant de traiter certains aspects des deux sous-typages précédents.

Sous-typage uniforme

Outil permettant de traiter certains aspects des deux sous-typages précédents.

Σ ne contient que des symboles de mêmes arités et polarités.
Un type est donc toujours un arbre infini, et une fonction totale $\{1, \ldots, k\}^* \to \Sigma$, où k est l'arité commune.
La polarité d'un nœud est indépendante de l'arbre.

Sous-typage uniforme

Outil permettant de traiter certains aspects des deux sous-typages précédents.

Σ ne contient que des symboles de mêmes arités et polarités.
Un type est donc toujours un arbre infini, et une fonction totale $\{1, \ldots, k\}^* \to \Sigma$, où k est l'arité commune.
La polarité d'un nœud est indépendante de l'arbre.

On se donne un ordre partiel $\leqslant_\Sigma$ sur Σ.

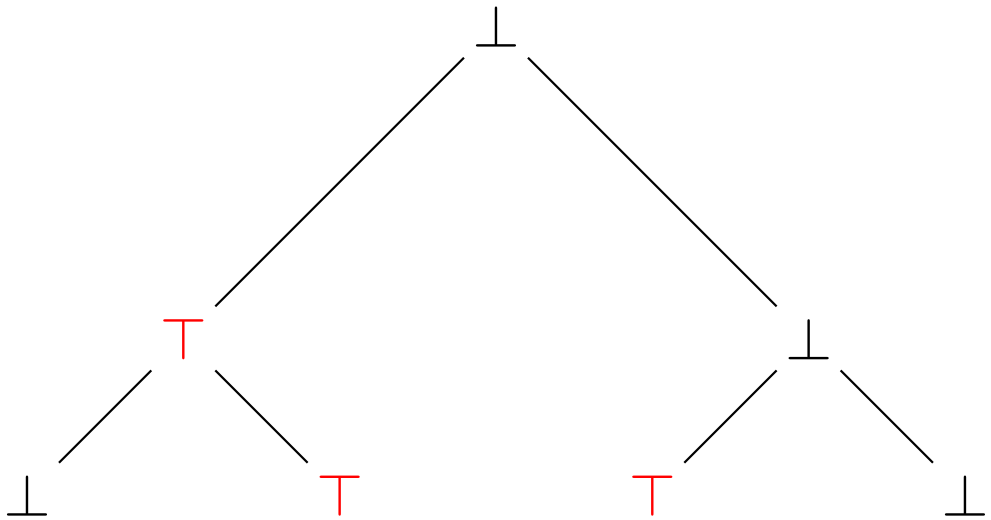
$$\tau_1 \leqslant \tau_2 \quad \text{ssi} \quad \forall \pi \in \{1, \ldots, k\}^*, \quad \tau_1(\pi) \leqslant_\Sigma^{\mathsf{pol}(\pi)} \tau_2(\pi)$$

Exemple

$\Sigma = \{\bot, \top\}$, $\bot \leqslant \top$, $k = 2$, $\mathrm{pol}(1) = -1, \mathrm{pol}(2) = 1$
Comme dans le cas du sous-typage non-structurel, il y a un plus petit type :

Exemple

$\Sigma = \{\bot, \top\}$, $\bot \leqslant \top$, $k = 2$, $\mathrm{pol}(1) = -1, \mathrm{pol}(2) = 1$
Comme dans le cas du sous-typage non-structurel, il y a un plus petit type :



Problèmes étudiés

Distinction syntaxe/sémantique

On se donne une signature Σ.

Distinction syntaxe/sémantique

On se donne une signature Σ.

Syntaxe d'une contrainte atomique :

$$C ::= x \leqslant y \mid x = f(x_1, \ldots, x_n)$$

Distinction syntaxe/sémantique

On se donne une signature Σ.

Syntaxe d'une contrainte atomique :

$$C ::= x \leqslant y \mid x = f(x_1, \ldots, x_n)$$

Sémantique d'une contrainte atomique :

Distinction syntaxe/sémantique

On se donne une signature Σ.

Syntaxe d'une contrainte atomique :

$$C ::= x \leqslant y \mid x = f(x_1, \ldots, x_n)$$

Sémantique d'une contrainte atomique :

- on se donne une fonction d'affectation σ, des interprétations de $=$ et de $\leqslant$

Distinction syntaxe/sémantique

On se donne une signature Σ.
Syntaxe d'une contrainte atomique :

$$C ::= x \leqslant y \mid x = f(x_1, \ldots, x_n)$$

Sémantique d'une contrainte atomique :

- on se donne une fonction d'affectation σ, des interprétations de $=$ et de $\leqslant$
- on interprète la valeur de vérité selon les règles :

$$\sigma \models x \leqslant y \iff \sigma(x) \leqslant \sigma(y)$$
$$\sigma \models x = f(x_1, \ldots, x_n) \iff \sigma(x) = f(\sigma(x_1), \ldots, \sigma(x_n))$$

Logique de contraintes du premier ordre

Logique de contraintes du premier ordre

Syntaxe :

$$\phi ::= C \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists x.\phi_1$$

Logique de contraintes du premier ordre

Syntaxe :

$$\phi ::= C \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists x.\phi_1$$

C'est suffisant.

Logique de contraintes du premier ordre

Syntaxe :

$$\phi ::= C \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists x.\phi_1$$

C'est suffisant.

Sémantique :

$$\sigma \models \phi_1 \wedge \phi_2 \iff \sigma \models \phi_1 \text{ et } \sigma \models \phi_2$$
$$\sigma \models \neg\phi_1 \iff \sigma \not\models \phi_1$$
$$\sigma \models \exists x.\phi_1 \iff \text{il existe } \tau \text{ tel que } \sigma[x := \tau] \models \phi_1$$

Résultats

Problème : étant donné une formule ϕ, a-t-on $\models \phi$?

Résultats

Problème : étant donné une formule ϕ, a-t-on $\models \phi$?

Ce problème se décline en trois versions :

Résultats

Problème : étant donné une formule ϕ, a-t-on $\models \phi$?

Ce problème se décline en trois versions :

- dans le cas du sous-typage structurel, c'est décidable quand on ne considère que des arbres finis [V. Kuncak et M. Rinard, 2003]

Résultats

Problème : étant donné une formule ϕ, a-t-on $\models \phi$?

Ce problème se décline en trois versions :

- dans le cas du sous-typage structurel, c'est décidable quand on ne considère que des arbres finis [V. Kuncak et M. Rinard, 2003]
- dans le cas non-structurel, c'est indécidable [Z. Su et al., 2002]

Résultats

Problème : étant donné une formule ϕ, a-t-on $\models \phi$?

Ce problème se décline en trois versions :

- dans le cas du sous-typage structurel, c'est décidable quand on ne considère que des arbres finis [V. Kuncak et M. Rinard, 2003]
- dans le cas non-structurel, c'est indécidable [Z. Su et al., 2002]
- dans le cas uniforme, c'est décidable ! [résultat du stage]

Types contraints

Généralisation des types polymorphes du système ML :

Types contraints

Généralisation des types polymorphes du système ML :

Soit double $\stackrel{\text{def}}{=} \lambda f, x \mapsto f(fx)$.
Le type de cette fonction dans le système ML est :

$$(\alpha \to \alpha) \to \alpha \to \alpha$$

Types contraints

Généralisation des types polymorphes du système ML :

Soit double $\stackrel{\text{def}}{=} \lambda f, x \mapsto f(fx)$.
Le type de cette fonction dans le système ML est :

$$(\alpha \to \alpha) \to \alpha \to \alpha$$

Mais double a aussi le type contraint plus général :

$$(\alpha \to \beta) \to \alpha \to \beta \backslash \{\beta \leqslant \alpha\}$$

Sous-typage de types contraints

La question

$$\alpha_1 \backslash C_1 \leqslant^? \alpha_2 \backslash C_2$$

où les deux types utilisent des ensembles de variables disjoints,

Sous-typage de types contraints

La question

$$\alpha_1 \backslash C_1 \leqslant^? \alpha_2 \backslash C_2$$

où les deux types utilisent des ensembles de variables disjoints, peut être exprimée par une formule logique du premier ordre de la forme :

$$\forall \alpha_2, x_1, \ldots, x_n.\, C_2 \Rightarrow \exists \alpha_1, y_1, \ldots, y_m.\, C_1 \Rightarrow \alpha_1 \leqslant \alpha_2$$

Sous-typage de types contraints

La question

$$\alpha_1 \backslash C_1 \leqslant^? \alpha_2 \backslash C_2$$

où les deux types utilisent des ensembles de variables disjoints, peut être exprimée par une formule logique du premier ordre de la forme :

$$\forall \alpha_2, x_1, \ldots, x_n.\, C_2 \Rightarrow \exists \alpha_1, y_1, \ldots, y_m.\, C_1 \Rightarrow \alpha_1 \leqslant \alpha_2$$

Le problème associé est donc décidable dans le cas structurel fini et le cas uniforme.

Sous-typage de types contraints

Mais le cas non-structurel reste toujours un problème ouvert, même pour une signature aussi simple que $\Sigma = \{\bot, f, \top\}$.

Sous-typage de types contraints

Mais le cas non-structurel reste toujours un problème ouvert, même pour une signature aussi simple que $\Sigma = \{\bot, f, \top\}$.

En essayant de le résoudre, on est tombé sur un algorithme d'approximation correct (mais pas complet).

Sous-typage de types contraints

Mais le cas non-structurel reste toujours un problème ouvert, même pour une signature aussi simple que $\Sigma = \{\bot, f, \top\}$.

En essayant de le résoudre, on est tombé sur un algorithme d'approximation correct (mais pas complet).

Idée : plonger les contraintes non-structurelles dans le monde uniforme où tout est décidable, puis transposer les résultats au monde non-structurel.

Implication de contraintes

En pratique, les types contraints peuvent être grands et compliqués. Il est donc important de pouvoir les simplifier. Cette simplification est liée au problème de l'*implication de contraintes* (*entailment* en anglais) :

Implication de contraintes

En pratique, les types contraints peuvent être grands et compliqués. Il est donc important de pouvoir les simplifier. Cette simplification est liée au problème de l'*implication de contraintes* (*entailment* en anglais) :
On dit que C *implique* $x \leqslant y$ si pour toute affectation σ telle que $\sigma \models C$, $\sigma \models x \leqslant y$.

Implication de contraintes

En pratique, les types contraints peuvent être grands et compliqués. Il est donc important de pouvoir les simplifier. Cette simplification est liée au problème de l'*implication de contraintes* (*entailment* en anglais) :
On dit que C *implique* $x \leqslant y$ si pour toute affectation σ telle que $\sigma \models C$, $\sigma \models x \leqslant y$.
Cela s'exprime en logique du premier ordre sous la forme :

$$\forall x_1, \ldots, x_n.\, C \Rightarrow x \leqslant y$$

C'est a priori plus simple,

Implication de contraintes

En pratique, les types contraints peuvent être grands et compliqués. Il est donc important de pouvoir les simplifier. Cette simplification est liée au problème de l'*implication de contraintes* (*entailment* en anglais) :
On dit que C *implique* $x \leqslant y$ si pour toute affectation σ telle que $\sigma \models C$, $\sigma \models x \leqslant y$.
Cela s'exprime en logique du premier ordre sous la forme :

$$\forall x_1, \ldots, x_n.\, C \Rightarrow x \leqslant y$$

C'est a priori plus simple, mais la situation est la même qu'avec le sous-typage de types contraints !

Implication de contraintes

En pratique, les types contraints peuvent être grands et compliqués. Il est donc important de pouvoir les simplifier. Cette simplification est liée au problème de l'*implication de contraintes* (*entailment* en anglais) :
On dit que C *implique* $x \leqslant y$ si pour toute affectation σ telle que $\sigma \models C$, $\sigma \models x \leqslant y$.
Cela s'exprime en logique du premier ordre sous la forme :

$$\forall x_1, \ldots, x_n.\, C \Rightarrow x \leqslant y$$

C'est a priori plus simple, mais la situation est la même qu'avec le sous-typage de types contraints !
...et on a le même algorithme.

Ressemblance entre non-structurel et uniforme

Il existe un lien très fort entre les deux théories :

Ressemblance entre non-structurel et uniforme

Il existe un lien très fort entre les deux théories :

- notre approximation est au moins aussi puissante que celle de V. Trifonov et S. Smith [1996], et la preuve de cela est exactement la preuve de la correction de leur algorithme

Ressemblance entre non-structurel et uniforme

Il existe un lien très fort entre les deux théories :

- notre approximation est au moins aussi puissante que celle de V. Trifonov et S. Smith [1996], et la preuve de cela est exactement la preuve de la correction de leur algorithme
- l'implication de contraintes uniformes est, comme dans le cas non-structurel, PSPACE-difficile, et la preuve est la même que celle de J. Rehof et F. Henglein [1998] dans le cas non-structurel

Récapitulatif et conclusion

- La motivation principale était la décidabilité de l'implication de contraintes et du sous-typage de types contraints.
- On a simplifié le problème en introduisant la notion de sous-typage uniforme, où tout est décidable.
- On a exhibé des algorithmes d'approximation utilisant cette notion, au moins aussi bons que ceux connus.
- Cela a révélé de nombreuses similarités entre le monde non-structurel et le monde uniforme.

Perspectives

- Les algorithmes ne sont pas complets : caractériser les contre-exemples.
- Donner une meilleure comparaison avec ceux connus.
- Une implémentation ?
- Adapter la théorie uniforme pour prouver la décidabilité de l'implication de contraintes ou du sous-typage de types contraints.

Questions ?