Le sous-typage d'OCaml

Jacques Garrigue

25 juin 2008

Objective Caml est équipé d'une relation de sous-typage. Pour éviter les interférences avec l'inférence de types, la règle de subsomption est explicite. Les conséquences de l' inférence ne s'arrêtent pas là. Les types peuvent contenir des variables, mais contrairement à $F_{\leq}$ dans lequel les variables sont accompagnées de contraintes de sous-typage, en OCaml les seules contraintes possibles sont des contraintes d'unification. Modulo cette différence, la relation de sous-typage est assez proche de celle de $F_{\leq}$.

Sous-typage

Nous allons définir la relation de sous-typage

$$E \vdash \tau_1 \leq \tau_2$$

E est un ensemble d'équations entre types. Cette relation est utilisée par la règle de subsomption:

$$\textsc{Subtype}\quad \frac{\Gamma \vdash e : \sigma(\tau_1) \quad E \vdash \tau_1 \leq \tau_2 \quad \sigma \models E}{\Gamma \vdash (e : \tau_1 :> \tau_2) : \sigma(\tau_2)}$$

où la notation $\sigma \models E$ signifie que σ est un unificateur de E. Du point de vue de l'inférence, les équations de E sont générées par l'algorithme de sous-typage et résolues par unification immédiatement après.

Règles de base

$$\textsc{Equal}\quad \frac{\tau_1 = \tau_2 \in E}{E \vdash \tau_1 \leq \tau_2} \qquad \textsc{Arrow}\quad \frac{E \vdash \tau_1' \leq \tau_1 \quad E \vdash \tau_2 \leq \tau_2'}{E \vdash l{:}\tau_1 \to \tau_2 \leq l{:}\tau_1' \to \tau_2'} \qquad \textsc{Tuple}\quad \frac{E \vdash \tau_1 \leq \tau_i' \ \ (1 \leq i \leq n)}{E \vdash \tau_1 \times \ldots \times \tau_n \leq \tau_1' \times \ldots \times \tau_n'}$$

La règle Equal joue le rôle de cas par défaut. Pour tirer un algorithme de ces règles, il suffit de lui donner la priorité la plus basse.

Le cas des constructeurs de type est plus général.

$$\textsc{Tconstr}\quad \frac{E \vdash \tau_i \leq \tau_i' \ (i \in P(t)) \quad E \vdash \tau_j' \leq \tau_j \ (j \in N(t)) \quad \tau_k = \tau_k' \in E \ (k \in I(t))}{E \vdash (\tau_1, \ldots, \tau_n)\, t \leq (\tau_1', \ldots, \tau_n')\, t}$$

où $P(t)$, $N(t)$ et $I(t)$ sont respectivement les positions des paramètres covariants, contravariants, et invariants de t. Ces ensembles sont disjoints, mais ne forment pas nécessairement une partition de $[1 \ldots n]$. En effet, certains paramètres peuvent ne pas apparaître dans la définition de t, et par conséquent disparaître lors du sous-typage.

Objets et variants

Les deux règles sont quasiment identiques si on utilise les définitions du polymorphisme structurel.

$$\textsc{Object}\quad \frac{U' \subseteq L \quad (l \mapsto \tau \in T, l \mapsto \tau' \in T')\ E \vdash \tau \leq \tau'}{E \vdash \langle L, U, T\rangle \leq \langle L', U', T'\rangle} \qquad \textsc{Variant}\quad \frac{U \subseteq L' \quad (l \mapsto \tau \in T, l \mapsto \tau' \in T')\ E \vdash \tau \leq \tau'}{E \vdash [L, U, T] \leq [L', U', T']}$$

L dénote les étiquettes requises, U les étiquettes admises. Dans le cas des objets, U ne peut être que L ou $\mathcal{L}$ (l'ensemble de toutes les étiquettes.)

Types récursifs

On a besoin d'une première extensions du jugement de sous-typage, pour mémoriser les relations qu'on est en train de prouver. Le paramètre S doit juste être propagé sans modification par les autres règles de sous-typage.

$$\textsc{RecSub}\quad \frac{\tau_1 \leq \tau_2 \in S}{E; S \vdash \tau_1 \leq \tau_2} \qquad \textsc{RecL}\quad \frac{E; S, (\tau_1 \text{ as } x) \leq \tau_2 \vdash \tau_1[(\tau_1 \text{ as } x)/x] \leq \tau_2}{E; S \vdash (\tau_1 \text{ as } x) \leq \tau_2}$$

$$\textsc{RecR}\quad \frac{E; S, \tau_1 \leq (\tau_2 \text{ as } x) \vdash \tau_1 \leq \tau_2[(\tau_2 \text{ as } x)/x]}{E; S \vdash \tau_1 \leq (\tau_2 \text{ as } x)}$$

Méthodes polymorphes

Pour pouvoir traiter le sous-typage entre méthodes polymorphes, on a cette fois-ci besoin de préfixer les groupes d'équations par des contraintes de bijection entre vecteurs de variables universelles. Autrement dit E est maintenant de la forme

$$E := \emptyset \mid E, \tau_1 = \tau_2 \mid E, \bar{u} \leftrightarrow \bar{v}.E$$

Pour tenir compte des variales universelles, on paramètrise la définition de l'unification par une bijection δ entre celles-ci: $\sigma \models_\delta E$. Ce paramètre agit de la façon suivante. Pour les équations simples, $\sigma \models_\delta \tau_1 = \tau_2$ (σ unifie τ_1 et τ_2 modulo δ) ssi $\delta(\sigma(\tau_1)) = \sigma(\tau_2)$. Pour les groupes préfixés, $\sigma \models_\delta \bar{u} \leftrightarrow \bar{v}.E$ (σ est un unificateur de $\bar{u} \leftrightarrow \bar{v}.E$ modulo δ ssi il existe une bijection δ' entre $\bar{u}$ et $\bar{v}$ telle que $\sigma \models_{\delta\circ\delta'} E$.

Comme les équations deviennent asymmétriques, il faut en théorie veiller à les permuter quand on descend dans un sous-terme contravariant (règles Arrow et Tconstr). Tout ceci est déjà traité correctement par l'unification.

$$\textsc{Poly}\quad \frac{E_1 \vdash \tau_1 \leq \tau_2 \quad \bar{u} \leftrightarrow \bar{v}.E_1 \in E}{E \vdash \forall \bar{u}.\tau_1 \leq \forall \bar{v}.\tau_2}$$

La règle Poly n'introduit pas d'instanciation. Il n'y a pas non plus sous-typage entre les bornes des variables universelles, contrairement à $F_{\leq}$, pour la bonne raison que les variables universelles n'ont pas de borne.

Extension: abbréviations privées

$$\textsc{Private}\quad \frac{E \vdash \tau[\tau_1 ... \tau_n/\alpha_1 ... \alpha_n] \leq \tau' \quad \texttt{type}\ (\alpha_1, ..., \alpha_m)\, t = \texttt{private}\ \tau}{E \vdash (\tau_1, \ldots, \tau_n)\, t \leq \tau'}$$

Cette règle peut entrer en conflit avec Tconstr. Dans l'algorithme de sous-typage il faudra donc lui donner une priorité inférieure à celle de Tconstr.

Extension: instanciation des méthodes

$$\frac{E \vdash \tau_1[\bar{\alpha}/\bar{u}] \leq \tau_2}{E \vdash \forall \bar{u}.\tau_1 \leq \forall.\tau_2} \textsc{InstPoly}$$

Les méthodes ont toujours des types universels, d'où le $\forall.\tau_2$ ci-dessus.

Propriétés

Comme on s'y attend, cette relation est réflexive et transitive. Mais pour prouver ces propriétés, il faut d'abord que E ait un unificateur.

Propriété 1 (réflexivité) *Pour tout τ, il existe E tel que $E \vdash \tau \leq \tau$ et $id \models E$.*

Propriété 2 (transitivité) *Si $E_1 \vdash \tau_1 \leq \tau_2$ et $E_2 \vdash \tau_2 \leq \tau_3$ et $\sigma \models E_1$ et $\sigma \models E_2$, avec $\mathrm{Img}(\sigma) \subset Var$ (càd. l'image d'une variable par σ est une variable), alors il existe E_3 tel que $E_3 \vdash \tau_1 \leq \tau_3$ et $\sigma \models E_3$.*

Une propriété importante vis à vis de l'inférence est la monotonie du sous-typage par rapport à la précision des types.

Propriété 3 (monotonie) *Si $E \vdash \tau_1 \leq \tau_2$, et si $\sigma' \circ \sigma \models E$, alors il existe E' tel que $E' \vdash \sigma(\tau_1) \leq \sigma(\tau_2)$ et $\sigma' \models E'$.*

On dispose bien sûr d'un algorithme complet pour vérifier le sous-typage.

Propriété 4 (complétude) *L'algorithme obtenu en partant de $?E; \emptyset; \emptyset \vdash \tau_1 \leq \tau_2$ et en appliquant les règles de dérivation du sous-typage, donnant les priorités les plus basses à* Equal *puis* Private*, et la plus haute à* RecSub*, est complet. C'est à dire que si il renvoie E, et si on peut prouver d'autre part $E'; \emptyset; \emptyset \vdash \tau_1 \leq \tau_2$ et $\sigma \models E'$, alors nécessairement $\sigma \models E$.*

L'algorithme decrit ci-dessus n'échoue jamais, grâce à la règle Equal qui retarde tout échec jusqu'à l'unification. On peut toutefois détecter certaines contradictions structurelles sans attendre l'unification; par exemple, le sous-typage entre deux n-uplets necessite qu'ils soient de la même arité.

L'énoncé de la complétude ci-dessus ne tient pas compte des variables introduites par InstPoly, qui peuvent avoir des noms différents dans des dérivations différentes, et entrainer que $\sigma \not\models E$. Pour traiter ce choix, il suffit d'ajouter une substitution spécifique: *il existe φ, telle que pour tout $\alpha \in fv(\tau_1, \tau_2)$, $\varphi(\alpha) = \alpha$, et $\sigma \circ \varphi \models E$.* La valeur de cette substitution n'influe pas sur le reste de l'inférence, puisque seuls $\sigma(\varphi(\tau_1)) = \sigma(\tau_1)$ et $\sigma(\varphi(\tau_2)) = \sigma(\tau_2)$ comptent.

Coercions simples

En plus du sous-typage explicite, on a une syntaxe pour le sous-typage semi-explicite.

$$\frac{\Gamma \vdash e : \sigma(enlarge(\tau))}{\Gamma \vdash (e :> \tau) : \sigma(\tau)} \textsc{Enlarge}$$

L'algorithme *enlarge* produit un sous-type de τ, qui possède τ comme instance.

Propriété 5 (expansion) *Pour tout type τ, il existe E tel que $E \vdash enlarge(\tau) \leq \tau$ et $id \models E$. De plus il existe σ tel que $\sigma(enlarge(\tau)) = \tau$.*

En théorie *enlarge* pourrait être l'identité. En pratique, on expanse les objets et variants fermés en position covariante, avec une limite sur la profondeur, pour éviter une explosion de la taille du type.