projet de compilation

Petit Rust
typage des ressources

version 1 — 30 novembre 2017

Pour le typage des ressources de Petit Rust, on se limite à la granularité des variables, comme indiqué dans le sujet. En particulier, on ne peut pas exprimer en Petit Rust l'emprunt d'un certain champ de structure, mais uniquement l'emprunt de la structure toute entière. Le principe du typage de ressources consiste à associer, en tout point du programme, un statut à chaque variable (visible à cet endroit-là). Ce statut est de trois natures différentes :

$$\mathit{statut} ::= \texttt{Vide} \mid \texttt{Plein} \mid \texttt{Emprunté}(m)$$

Le statut `Vide` signifie que le contenu de la variable a été déplacé et ne peut plus être utilisé (en revanche, il peut être remplacé par autre chose). Le statut `Plein` signifie que la variable est le propriétaire de la ressource ; on peut faire ce que l'on veut avec. Enfin, le statut $\texttt{Emprunté}(m)$ signifie que le contenu de la variable a été emprunté, de façon mutable si $m = \texttt{mut}$.

Dans Petit Rust, une *durée de vie* coïncide avec un bloc, un appel étant considéré comme se faisant dans un nouveau bloc implicite (c'est le bloc qui correspond au corps de la fonction appelée) [1]. On note les durées de vie α, β, etc. On introduit une relation d'ordre partiel sur les durées de vie, notée $\alpha \leq \beta$, signifiant « la durée de vie β contient la durée de vie α ».

Pendant le typage des ressources, on va associer à chaque expression un type enrichi par des durées de vie au niveau de chaque emprunt, c'est-à-dire un type de la forme

$$\tau ::= \texttt{()} \mid \texttt{i32} \mid \texttt{bool} \mid S \mid \texttt{Vec<}\tau\texttt{>} \mid \texttt{\&}^{\alpha} m\, \tau$$

On introduit le jugement $\vdash_\alpha \tau$ qui signifie « le type τ est valide pour la durée de vie α » et on le définit ainsi :

$$\frac{\tau \in \{\texttt{()}, \texttt{i32}, \texttt{bool}, S\}}{\vdash_\alpha \tau} \qquad \frac{\vdash_\alpha \tau}{\vdash_\alpha \texttt{Vec<}\tau\texttt{>}} \qquad \frac{\vdash_\beta \tau \quad \beta \geq \alpha}{\vdash_\alpha \texttt{\&}^{\beta} m\, \tau}$$

On doit notamment vérifier que le type de chaque sous-expression est bien valide pour la durée de vie courante, *i.e.*, du bloc dans lequel il se trouve. Dans Petit Rust, un emprunt $\texttt{\&}m\, e$ se fait toujours à une durée de vie égale au bloc courant.

La relation d'ordre sur les durées de vie induit une relation d'ordre sur les types, notée $\tau_1 \leq \tau_2$, de la manière suivante :

$$\frac{}{\tau \leq \tau} \qquad \frac{\tau \leq \tau'}{\texttt{Vec<}\tau\texttt{>} \leq \texttt{Vec<}\tau'\texttt{>}} \qquad \frac{\tau \leq \tau' \quad \alpha \geq \beta}{\texttt{\&}^{\alpha}\, \tau \leq \texttt{\&}^{\beta}\, \tau'} \qquad \frac{\alpha \geq \beta}{\texttt{\&}^{\alpha}\texttt{mut}\, \tau \leq \texttt{\&}^{\beta}\texttt{mut}\, \tau}$$

Lors d'une affectation e_1=e_2, on doit notamment vérifier qu'on a bien $\tau_2 \leq \tau_1$, où τ_2 est le type de e_2 et τ_1 le type de e_1. Attention : on parle ici des types obtenus *après* les éventuelle coercions de $\texttt{\&mut}\, \tau$ vers $\texttt{\&}\, \tau$ décrites dans le sujet du projet.

Il y a des vérifications à faire à plusieurs endroits, notamment

— lorsqu'une valeur gauche est utilisée comme une valeur droite (en particulier le statut d'une variable peut être modifié) ;

1. Une durée de vie peut donc être matérialisée par un entier, qui compte un nombre de blocs imbriqués sous lesquels on se trouve.

— lors d'une affectation e_1=e_2 : outre la vérification de sous-typage ci-dessus, il faut aussi vérifier que la ressource correspondant à la valeur gauche e_1 n'est pas empruntée ;
— lors d'un emprunt, c'est-à-dire pour la construction `&`$m\,e$. On sait que e est une valeur gauche. Si e est de la forme $e_1[e_2]$ ou $e_1.x$, on va chercher la variable concernée dans e_1. Si e est de la forme $*e_1$, on pourra imposer que e_1 soit alors une variable.

Remerciements. Merci à Jacques-Henri Jourdan pour son aide dans la préparation de ce document.