Traduction des langages

Typage

Objectif :
- Définir la nouvelle structure d'arbre obtenue après la passe de typage
- Définir les actions à réaliser par la passe de typage

1 Rappel : le typage du langage Rat

1.1 Grammaire du langage RAT

1. $PROG' \rightarrow PROG\$$
2. $PROG \rightarrow FUN\ PROG$
3. $FUN \rightarrow TYPE\ id\ (\ DP\ )\ BLOC$
4. $PROG \rightarrow\ id\ \ BLOC$
5. $BLOC \rightarrow\ \{\ IS\ \}$
6. $IS \rightarrow\ I\ \ IS$
7. $IS \rightarrow \Lambda$
8. $I \rightarrow\ TYPE\ \ id\ =\ E\ ;$
9. $I \rightarrow\ id\ =\ E\ ;$
10. $I \rightarrow\ const\ \ id\ =\ entier\ ;$
11. $I \rightarrow\ print\ \ E\ ;$
12. $I \rightarrow\ if\ \ E\ \ BLOC\ \ else\ \ BLOC$
13. $I \rightarrow\ while\ \ E\ \ BLOC$
14. $I \rightarrow\ return\ \ E\ ;$
15. $DP \rightarrow \Lambda$
16. $DP \rightarrow TYPE\ id\ DP$
17. $TYPE \rightarrow\ bool$
18. $TYPE \rightarrow\ int$
19. $TYPE \rightarrow\ rat$
20. $E \rightarrow call\ id\ (\ CP\ )$
21. $CP \rightarrow \Lambda$
22. $CP \rightarrow E\ CP$
23. $E \rightarrow\ [\ E\ /\ E\ ]$
24. $E \rightarrow\ num\ \ E$
25. $E \rightarrow\ denom\ \ E$
26. $E \rightarrow\ id$
27. $E \rightarrow\ true$
28. $E \rightarrow\ false$
29. $E \rightarrow\ entier$
30. $E \rightarrow\ (\ E\ +\ \ E\ )$
31. $E \rightarrow\ (\ E\ *\ \ E\ )$
32. $E \rightarrow\ (\ E\ =\ \ E\ )$
33. $E \rightarrow\ (\ E\ <\ \ E\ )$
34. $E \rightarrow\ (\ E\ \ )$

1.2 Jugements de typage du langage RAT

Axiomes

— $\dfrac{x \in \sigma \qquad \sigma(x) = \tau}{\sigma \vdash x : \tau}$

Si x a été ajouté plusieurs fois à σ, $\sigma(x)$ renvoie le dernier type associé à x.

— $\sigma \vdash true : bool$

— $\sigma \vdash false : bool$

— $\sigma \vdash entier : int$

Expression

— $\dfrac{\sigma \vdash E_1 : int \qquad \sigma \vdash E_2 : int}{\sigma \vdash [\ E_1\ /\ E_2\ ] : rat}$

— $\dfrac{\sigma \vdash E : rat}{\sigma \vdash num\ E : int}$

— $\dfrac{\sigma \vdash E : rat}{\sigma \vdash denom\ E : int}$

— $\dfrac{\sigma \vdash E : \tau}{\sigma \vdash (E) : \tau}$

— $\dfrac{\sigma \vdash E_1 : int \quad \sigma \vdash E_2 : int}{\sigma \vdash (E_1 + E_2) : int}$

— $\dfrac{\sigma \vdash E_1 : rat \quad \sigma \vdash E_2 : rat}{\sigma \vdash (E_1 + E_2) : rat}$

— $\dfrac{\sigma \vdash E_1 : int \quad \sigma \vdash E_2 : int}{\sigma \vdash (E_1 * E_2) : int}$

— $\dfrac{\sigma \vdash E_1 : rat \quad \sigma \vdash E_2 : rat}{\sigma \vdash (E_1 * E_2) : rat}$

— $\dfrac{\sigma \vdash E_1 : int \quad \sigma \vdash E_2 : int}{\sigma \vdash (E_1 = E_2) : bool}$

— $\dfrac{\sigma \vdash E_1 : bool \quad \sigma \vdash E_2 : bool}{\sigma \vdash (E_1 = E_2) : bool}$

— $\dfrac{\sigma \vdash E_1 : int \quad \sigma \vdash E_2 : int}{\sigma \vdash (E_1 < E_2) : bool}$

— On se limitera à ces signatures.

Structures de contrôle

— $\dfrac{\sigma \vdash E : bool \qquad \sigma, \tau_r \vdash BLOC_1 : void \qquad \sigma, \tau_r \vdash BLOC_2 : void}{\sigma, \tau_r \vdash if\ E\ BLOC_1\ else\ BLOC_2 : void, []}$

— $\dfrac{\sigma \vdash E : bool \qquad \sigma, \tau_r \vdash BLOC : void}{\sigma, \tau_r \vdash while\ E\ BLOC : void, []}$

Déclaration / affectation

— $\dfrac{\sigma \vdash TYPE : \tau \qquad \sigma \vdash E : \tau}{\sigma, \tau_r \vdash TYPE\ id = E : void, [id, \tau]}$

— $\dfrac{\sigma \vdash id : \tau \qquad \sigma \vdash E : \tau}{\sigma, \tau_r \vdash id = E : void, []}$

Autres instructions

— $\dfrac{}{\sigma, \tau_r \vdash const\ \ id\ \ =\ \ entier : void, [id, int]}$

— $\dfrac{\sigma \vdash E : \tau}{\sigma, \tau_r \vdash print\ \ E : void, []}$

Déclaration de fonction

— $\dfrac{A \quad B \quad C}{\sigma \vdash TYPE\ id\ (DP)\ BLOC : void, [id, \tau_p \to \tau_r]}$

— A : $\sigma \vdash TYPE : \tau_r$
— B : $\sigma \vdash DP : \tau_p, \sigma_p \qquad (DP = TYPE_1\ id_1 \ldots TYPE_n\ id_n)$
— C : $(id, \tau_p \to \tau_r) :: \sigma_p @ \sigma, \tau_r \vdash BLOC : void$

— $\dfrac{\sigma \vdash TYPE_1 : \tau_1 \quad \ldots \quad TYPE_n : \tau_n}{\sigma \vdash TYPE_1\ id_1\ \ldots\ TYPE_n\ id_n : \tau_1 \times \cdots \times \tau_n, [(id_1, \tau_1); \ldots; (id_n, \tau_n)]}$

Appel de fonction

— $\dfrac{\sigma \vdash id : \tau_1 \to \tau_2 \qquad E_1 \ \ldots \ E_n : \tau_1}{\sigma \vdash call\ id\ (E_1 \ \ldots \ E_n) : \tau_2}$

— $\dfrac{\sigma \vdash E_1 : \tau_1 \quad \ldots \quad E_n : \tau_n}{\sigma \vdash E_1 \ \ldots \ E_n : \tau_1 \times \cdots \times \tau_n}$

Retour de fonction

— $\dfrac{\sigma \vdash E : \tau_r}{\sigma, \tau_r \vdash return\ E : void, []}$

Suite d'instructions

— $\dfrac{\sigma, \tau_r \vdash IS : void, \sigma'}{\sigma, \tau_r \vdash \{IS\} : void} \qquad \dfrac{\sigma, \tau_r \vdash IS : void, \sigma'}{\sigma, \tau_r \vdash BLOC : void}$ ($BLOC$ et $\{IS\}$ sont la même chose)

— $\dfrac{\sigma, \tau_r \vdash I : void, \sigma' \qquad \sigma' @ \sigma, \tau_r \vdash IS : void, \sigma''}{\sigma, \tau_r \vdash I\ IS : void, \sigma'' @ \sigma'}$

— $\sigma, \tau_r \vdash \ \ : void$

Le programme

— $\dfrac{\sigma \vdash FUN : void, \sigma' \qquad \sigma' @ \sigma \vdash PROG : void, \sigma''}{\sigma \vdash FUN\ PROG : void, \sigma'' @ \sigma'}$

— $\dfrac{\sigma, void \vdash BLOC : void}{\sigma \vdash id\ BLOC : void}$

2 Passe de typage

Nous rappelons qu'un compilateur fonctionne par passes, chacune d'elle réalisant un traitement particulier (gestion des identifiants, typage, placement mémoire, génération de code,...). Chaque passe parcourt, et potentiellement modifie, l'AST.

La seconde passe est une passe de typage. C'est elle qui vérifie la conformité des types déclarés et associe aux identifiants leurs informations de type.

2.1 Structure de l'AST après la passe de typage

La passe de typage réalise des vérifications de type qui nécessitent une mise à jour des informations de type de identificateurs. Elle prépare également les passes suivantes, par exemple en choisissant la "version" de l'opérateur à utiliser en cas de surcharge des opérateurs.

▷ **Exercice 1** *Définir la structure de l'AST après la passe de typage.*

2.2 Actions à réaliser lors de la passe de typage

▷ **Exercice 2** *Définir les actions à réaliser lors de la passe de typage.*