Compilation

TP 3 : Types

C. Alias & G. Iooss

Exercice 1. *Construction des types*

Récupérez le fichier `dcc_types.tgz` et décompactez le dans votre répertoire.

- **Inspectez les fichiers Type.h/.cc**. Dans le main, écrire le code pour créer et afficher le type `char[8]`.
- **Ouvrez le fichier parser.ypp**. Quel est l'attribut du non-terminal `type`? Complétez les règles de `type` pour construire convenablement les types.
- **Inspectez les fichiers SymbolTable.h/.cc**. Dans `parser.ypp`, à quoi sert `add_type($3,$2)` (aprés la ligne 215 - règle pour `type_def`)? Ajoutez `print_symbols(cout)` pour afficher les types enregistrés dans la table des symboles (Note: Au lieu de le faire dans le `main.cc`, faites-le dans la règle `prog` avant de normaliser les types). Testez sur `tests/test.c`.

Exercice 2. *Normalisation et types bien formés*

- `Type` comporte une méthode `print_dot()` qui produit la représentation `dotty` (graphe) du type courant (`dot -Tps test.dot > test.ps`). Expérimentez.
- Les types comportent encore des identificateurs, qu'il reste à remplacer par leur définition. Cette étape est appelée *normalisation* et se réalise après le dernière réduction de `type_def_list` (dernière ligne de `parser.ypp`).
- **Inspectez le code de normalize_types (SymbolTable.cc)**. Affichez le graphe (print_dot) de `list_t` normalisé.
- **Inspectez le code de is_well_formed (Type.cc)**. Après cette étape, on a la garantie que les types sont tous bien formés.
- Que fait reset_functions() (parser.ypp, dernière ligne)?

Exercice 3. *Equivalence de types*

Avant de contrôler les fonctions, il nous faut une équivalence entre types. Allez dans `Type.cc` (ligne 116), et **implémentez l'équivalence de types**.

Exercice 4. *Contrôle de type*

Chaque fois qu'une fonction est déclarée (parser.ypp, ligne 478), sa signature est ajoutée à la table des symboles. `add_function()` créé une nouvelle (signature de) fonction courante. Puis `add_argument_type()` ajoute les types des arguments. `add_argument()` déclare un argument et `add_local_var` déclare une nouvelle variable locale. Ces informations sont ensuite utilisées pour typer les expressions dans le corps de la fonction.

- **Inspectez** les règles de `function`, `declare_args`, `declare_local_vars`.
- Il est temps de contrôler les types... Inspectez les règles de `stmt`. Comment le polymorphisme de l'affectation est il traité?
- **Inspectez** la règle du return.
- **Inspectez** l'appel de procedure (fonction qui retourne void). À quoi correspond arg_type? Que fait type_check()?
- **Complétez** les règles du non-terminal `expr` pour **contrôler les types**.