Sémantique et Traduction des Langages
Epreuve de contrôle – Session 1 et 3
Avec documents de cours – Sans questions
Mercredi 15 Mai
Durée : 1h30
Sujet à rendre avec votre copie
Nom :
Prénom :

Les calculatrices, ordinateurs et téléphones portables sont interdits.

N.B. : Vous attacherez la plus grande importance à la clarté, à la précision et à la concision de la rédaction.
Toute réponse devra être justifiée.
Si vous repérez ce qui vous semble être une erreur d'énoncé, signalez celle-ci sur votre copie et poursuivez la composition en expliquant les hypothèses que vous avez faites.
Il est conseillé de lire tout le texte avant de répondre, pour avoir une vision globale de l'épreuve et traiter les questions dans l'ordre correspondant le mieux à vos connaissances.

1 Questions de cours

Vous donnerez des réponses précises et concises en prenant des exemples concrets si cela est nécessaire pour illustrer votre propos.

a/ Définir le rôle des étapes d'analyses lexicale, syntaxique et sémantique en précisant quelles sont les informations échangées entre ces étapes ? Préciser comment sont spécifiés les analyseurs lexicaux et syntaxiques.

b/ Quels sont les objectifs principaux de la spécification formelle de la sémantique d'un langage ?

c/ Quelles sont les différences entre des attributs hérités et synthétisés dans une grammaire attribuée ?

d/ En utilisant les règles de sémantique opérationnelle de miniML étudiée en cours, TD et TP, construire l'arbre correspondant à l'xécution de l'expression suivante (il s'agit d'appliquer les règles de sémantique formelle pour calculer la valeur – forme similaire à la construction des preuves en déduction naturelle dans le cours de Modélisation) :

$$\{\texttt{i} \mapsto 1\} \vdash (\texttt{function j} -> \texttt{i} + \texttt{j})\ (2) \Rightarrow 3$$

2 Sémantique attribuée

L'objectif est de définir une sémantique attribuée pour calculer la valeur d'une fraction rationnelle définie par la grammaire (A, V, F, P) avec l'alphabet $A = \{\texttt{c}, \texttt{d}, \texttt{m}\}$ (`c` désigne un chiffre, `d` désigne la division `/` et `m` désigne l'opposé `-`), les non-terminaux $V = \{F, N, S\}$, l'axiome F et les règles de production :

$$P = \left\{ \begin{array}{l} F \rightarrow N\,\texttt{d}\,N \\ F \rightarrow \texttt{m}\,N\,\texttt{d}\,N \\ N \rightarrow \texttt{c}\,S \\ S \rightarrow \Lambda \\ S \rightarrow \texttt{c}\,S \end{array} \right\}$$

a/ Construire l'arbre syntaxique (arbre de dérivation) pour l'exemple `-1/16`.

b/ Proposer des attributs sémantiques pour les non terminaux de V et décorer l'arbre de dérivation donné à la question précédente (vous ne recopierez pas l'arbre) pour calculer la valeur de la fraction rationnelle `-1/16`.

c/ Définir les actions sémantiques pour les règles de production P qui calculent la valeur de la fraction rationnelle.

3 Traitement par cas en miniML

L'objectif est de compléter le langage miniML étudié en cours, TD et TP par l'expression `match` qui calcule la valeur d'une expression puis compare celle-ci avec différents cas pour décider de l'expression suivante à évaluer. Si aucun des cas ne correspond à cette valeur, l'expression par défaut sera l'expression suivante. Ensuite, elle renvoie la valeur de cette expression suivante. Pour simplifier les choses, nous nous plaçons dans le cadre de la version de miniML sans effets de bord.

Le programme suivant est de type `int` et renvoie la valeur 1.

```
match (1+2) with
  | 1 -> 0  (* Cas valeur 1 *)
  | 3 -> 1  (* Cas valeur 3 *)
  | _ -> -1 (* Expression par défaut *)
```

La syntaxe de miniML est étendue de la manière suivante :

$$\begin{array}{rcl} Expr & \rightarrow & \cdots \\ & | & \texttt{match}\ Expr\ \texttt{with}\ |\ Val\ \texttt{->}\ Expr\ \ldots\ |\ Val\ \texttt{->}\ Expr |\ \_\ \texttt{->}\ Expr \end{array}$$

a/ Ecrire les règles de sémantique opérationnelle sans effets de bord pour cette nouvelle expression pour les cas sans erreurs. Les jugements seront de la forme : $\gamma \vdash \texttt{match}\ e\ \texttt{with}\ |\ v_1\ \texttt{->}\ e_1\ \ldots\ |\ v_n\ \texttt{->}\ e_n |\ \_\ \texttt{->}\ e_d \Rightarrow v$

b/ Ecrire les règles de sémantique opérationnelle sans effets de bord pour les cas d'erreur correspondant à cette nouvelle expression. Les jugements seront de la forme : $\gamma \vdash \texttt{match}\ e\ \texttt{with}\ |\ v_1\ \texttt{->}\ e_1\ \ldots\ |\ v_n\ \texttt{->}\ e_n |\ \_\ \texttt{->}\ e_d \Rightarrow \bot$

c/ Ecrire les règles de typage pour cette nouvelle expression. Les jugements seront de la forme : $\sigma \vdash \texttt{match}\ e\ \texttt{with}\ |\ v_1\ \texttt{->}\ e_1\ \ldots\ |\ v_n\ \texttt{->}\ e_n |\ \_\ \texttt{->}\ e_d : \tau$

Le type CaML `ast` est étendu pour prendre en compte cette nouvelle expression :

```
type ast =
| MatchNode of ast * ((ast * ast) list) * ast
| ...
```

La fonction de typage `type_of_expr` est étendue par le cas :

```
let rec type_of_expr expr env =
  match expr with
    | (MatchNode selection choix defaut) -> ruleMatch selection choix defaut env
    | ...
```

d/ Proposer une définition en CaML pour la fonction `ruleMatch`.

La fonction d'évaluation `value_of_expr` est étendue par le cas :

```
let rec value_of_expr expr env =
  match expr with
    | (MatchNode selection choix defaut) -> ruleMatch selection choix defaut env
    | ...
```

e/ Proposer une définition en CaML pour la fonction `ruleMatch`.

4 Instruction de choix en Bloc

L'objectif est de compléter le langage Bloc par une instruction de choix selon la valeur d'une expression. Nous ajoutons pour cela les instructions `select` et `case` inspirées des langages de la famille C (C, C++, C#, Java, etc) selon la syntaxe CUP suivante :

```
Instruction ::= UL_Select UL_Parenthese_Ouvrante Expression:valeur UL_Parenthese_Fermante
                UL_Accolade_Ouvrante ListeChoix:choix Defaut:defaut UL_Accolade_Fermante
     {:




     :} ;

ListeChoix ::= Choix:choix ListeChoix:reste
     {:




     :}
           |
     {:




     :} ;

Choix ::= UL_Case Valeur:valeur UL_Deux_Points Bloc:corps
     {:




     :} ;

Defaut ::= UL_Default UL_Deux_Points Bloc
     {:




     :}
           |
     {:




     :} ;
```

Contrairement à la sémantique des langages de la famille C (C, C++, C#, Java, etc), lors de la fin de l'exécution du bloc associé à un choix, le flot de contrôle est transféré à la fin de l'instruction de sélection.

Le type de l'expression sur laquelle la sélection est effectuée doit être compatible avec le type des valeurs associées aux choix.

Les valeurs associées aux choix doivent être de type compatible avec les entiers, les caractères, les booléens ou les chaînes de caractères.

Le programme suivant ne contient pas d'erreurs et s'arrête en affichant la valeur 1.

```
test {
  int i = 0;
  switch (i+1) {
    case 0 : {
      print 0;
    }
    case 1 : {
      print 1;
    }
    case 5 : {
      print 5;
    }
    default : {
      print -1;
    }
  }
}
```

Proposer un programme TAM qui pourrait être généré à partir de ce programme Bloc.

4.1 Vérification et construction de l'arbre abstrait

a/ Proposer sans les détailler une ou plusieurs classes pour représenter cette nouvelle instruction dans l'arbre abstrait.
b/ Modifier la grammaire CUP de Bloc pour construire l'arbre abstrait correspond à cette instruction.

4.2 Gestion de la table des symboles

a/ Expliquer quelles sont les traitements nécessaires pour gérer la table des symboles de cette nouvelle instruction.
b/ Proposer en Java des méthodes de gestion de la table des symboles de cette/ces classe(s).

4.3 Typage

a/ Expliquer quelles sont les traitements nécessaires pour gérer le typage de cette nouvelle instruction.
b/ Proposer en Java des méthodes de typage de cette/ces classe(s).

4.4 Génération de code

a/ Expliquer quelles sont les traitements nécessaires pour gérer la génération de code pour cette nouvelle instruction.
b/ Proposer en Java des méthodes de génération de code pour cette/ces classe(s).