UFR 919 Informatique
Master Informatique – Spécialité STL
UE ILP

TD/TME3 — Aspects statiques et dynamiques en ILP

Est qualifié de *statique* ce que l'on peut dire d'un code avant son exécution (par exemple pendant la phase de compilation) ; est qualifié de *dynamique* ses effets que l'on ne peut décider qu'à l'exécution. Une propriété statique est décidable par analyse de l'AST. Attention, certaines propriétés peuvent dans certains langage être statiques alors qu'elles sont dynamique dans d'autres. Par exemple : le typage. En ILP, le typage est dynamique.

1 Statique ou dynamique ?

Nous allons donner quelques exemples de programmes (en syntaxe xml ou allégée) ainsi que le résultat de leur analyse (génération de l'AST) ou de leur exécution. Lorsqu'il s'agira d'une erreur, vous direz si elle intervient lors de l'analyse ou de l'exécution. Lorsqu'il s'agira d'un résultat vous expliquerez s'il était prévisible ou non au seul vu du code.

Les résultats des évaluations seront donnés après le mot `Value :` et les affichages seront donnés après `Ouput :[`.

1.1 Typage

Soit

```
<programme1>
  <entier valeur="quarantedeux"/>
</programme1>
```

on obtient

```
Exception in thread "main" fr.upmc.ilp.ilp1.eval.EASTException:
java.lang.NumberFormatException: For input string: "qua"
```

Soit

```
<programme1>
<operationBinaire operateur ="==">
<operandeGauche><booleen valeur="false"/></operandeGauche>
<operandeDroit><entier valeur="0"/></operandeDroit>
</operationBinaire>
</programme1>
```

on obtient

```
Exception in thread "main" fr.upmc.ilp.ilp1.runtime.EvaluationException: ==:
Argument 1 is not number
```

Soit

```
<programme1>
<operationBinaire operateur ="+">
<operandeGauche>
<operationBinaire operateur="|">
<operandeGauche><booleen valeur="true"/></operandeGauche>
<operandeDroit><entier valeur="10"/></operandeDroit>
</operationBinaire>
</operandeGauche>
<operandeDroit><entier valeur="5"/></operandeDroit>
</operationBinaire>
</programme1>
```

on obtient

```
Exception in thread "main" fr.upmc.ilp.ilp1.runtime.EvaluationException: +:
Argument 1 is not number
```

Soit

```
<programme1>
<operationBinaire operateur ="+">
<operandeGauche>
<operationBinaire operateur="|">
<operandeGauche><booleen valeur="false"/</operandeGauche>
<operandeDroit><entier valeur="10"/></operandeDroit>
</operationBinaire>
</operandeGauche>
<operandeDroit><entier valeur="5"/></operandeDroit>
</operationBinaire>
</programme1>
```

on obtient

```
Value: 15
```

Soit

```
<programme1>
<operationBinaire operateur ="+">
<operandeGauche><flottant valeur="4.73"/></operandeGauche>
<operandeDroit><entier valeur="10"/></operandeDroit>
</operationBinaire>
</programme1>
```

on obtient

```
Value: 14.73
```

1.2 Liaison de variables, environnement d'évaluation

On adoptera ici une synatxe «à la Python/ML» (l'indentation détermine les blocs) sauf dans un ou deux cas.

Soit

```
let x1 = 10 in
  let x2 = x1 + 20 in
    let x1 = 40 in
      print x1
      print x2
    print x1
    print x2
```

on obtient

```
Output:[
40301030
]
```

Soit

```
<programme1>
<blocUnaire>
  <variable nom="print"/>
  <valeur><chaine>print</chaine></valeur>
  <corps>
    <invocationPrimitive fonction="print">
      <variable nom="print"/>
    </invocationPrimitive>
  </corps>
</blocUnaire>
</programme1>
```

on obtient

```
Output:[
fr.upmc.ilp.ilp1.runtime.PrintStuff$PrintPrimitive@3e89c3
]
```

Soit

```
let pi = "le nombre pi:" in
  print pi
print pi
```

on obtient

```
Output:[
le nombre pi:3.141592653589793
]
```

Soit

```
let e = 2.7182818 in
  print e
print e
```

on obtient

```
Exception in thread "main" fr.upmc.ilp.ilp1.runtime.EvaluationException:
Variable sans valeur: e
```

2 Aspect dynamique : une nouvelle primitive

Le caractère dynamique tient à l'indétermination de certaines valeurs. On obtient une cause simple d'indétermination en introduisant une *primitive de lecture*.

Il est facile d'introduire en ILP1 une primitive de lecture en s'inspirant de ce qui est fait pour les affichages. Une primitive de lecture pour ILP1 ne peut pas être définie en ILP1 : elle a sa place dans la bibliothèque d'exécution (*runtime*).

Pour simplifier l'implantation, les données lues par un programme ILP1 le seront sur la ligne de commande (argument des méthodes `main` en Java). Chaque invocation de la primitive récupèrera un argument supplémentaire de la ligne de commande (ignorer le premier qui est le nom du programme).

Sur le modèle de `PrintStuff`, définissez une classe `ReadStuff` implantant
– un constructeur `ReadStuff (String[] inputArray)`
– une méthode pour l'extension de l'environnement initial `public ILexicalEnvironment extendWithReadPrimitives (final ILexicalEnvironment lexenv)`
– une méthode définissant l'évaluation de la primitive `private class ReadIntPrimitive extends AbstractInvokableImpl`
Puis, sur le modèle de `fr.upmc.ilp.ilp1.eval.EASTFileTest`, créer une procédure permettant d'évaluer des programmes ILP1 utilisant cette nouvelle primitive. Appelez `TME3EASTFileTest` cette nouvelle version. Installez le tout dans un paquetage séparé : `fr.upmc.ilp.ilp1tme3`.

3 Analyse statique

3.1 Taille de l'environnement

Dans notre version primaire d'ILP, il est facile de connaître par avance la taille de l'environnement nécessaire à l'évaluation d'un programme ILP.

– Qu'est ce qui détermine cette taille ?

– Cette propriété est-elle vraie de tous les langages ?

Le calcul de la taille de l'environnement peut être fait sur l'AST. En conséquence, on peut étendre les classes `EAST*` qui définissent les constructions du langage ILP1 en des classes `TME3EAST*` qui implantent la méthode de calcul de la taille. On procèdera par extension et implantation d'une interface `IEnvironmentSize` donnant la signature de la méthode de calcul.

On a ainsi définit une nouvelle version des AST : il faut définir une nouvelle *fabrique* pour ceux-ci (disons, `TME3EASTFactory`).

Enfin, modifiez la classe `TME3EASTFileTest` de façon à invoquer le calcul de la taille et l'afficher.

3.2 Accès direct à l'environnement

Une autre donnée statique est la position de la valeur d'une variable dans l'environnement : son indice par rapport à la dernière valeur introduite.

Attention : une *même variable* peut avoir des positions différentes selon son *occurence* ; c'est-à-dire selon l'endroit où elle est utilisée.

Soit

```
let x = 123 in
  let y = x + 456 in
    print x + y
  print x
```

donnez pour chacune de ces lignes l'état de l'environnement et, pour chaque utilisation de `x` sa position dans celui-ci.

Pour calculer la position/indice de chaque occurence de variable, on simule le processus d'évaluation. Intuitivement, on parcours l'arbre de syntaxe abstraite en maintenant à jour une liste des variables déclarées.

– Comment cette liste évolue-t-elle ?

On rajoute cette nouvelle fonctionnalité aux `TME3EAST*` par l'implantation de deux nouvelles méthodes
– `public void setIndexes(List<String> varList)` pour le calcul et la mise à jour des indices des variables ;
– `public void printIndexes()` pour l'affichage des valeurs des indices des variables utilisées dans le programme ILP.
Selon la méthode utilisez ci-dessus (extension/implantation), implantez et utilisez sur des exemples de programmes ILP cette fonctionnalité.