

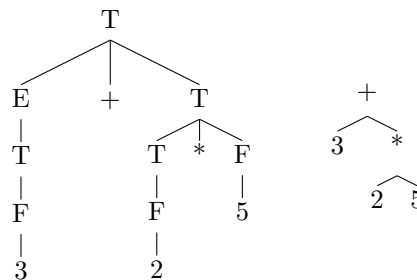
## TP02 - CONSTRUCTION DE L'ARBRE ABSTRAIT

### 1. OBJECTIF

L'objectif de ce TP est de construire un arbre abstrait correspondant au programme en langage source. Cet arbre abstrait sera utilisé lors des étapes suivantes de la compilation, en particulier pour la construction des tables de symboles et de la production du code trois adresses.

### 2. ARBRE ABSTRAIT

Un arbre abstrait constitue une représentation d'un programme plus simple et plus naturelle que l'arbre de dérivation pour la suite du processus de compilation. Il ne garde de la structure syntaxique que les parties nécessaires à l'analyse sémantique et à la production de code. En guise d'illustration, voici un arbre de dérivation et l'arbre abstrait lui correspondant.



### 3. CONSTRUCTION DE L'ARBRE ABSTRAIT

L'arbre abstrait peut être construit directement lors de l'analyse syntaxique, en ajoutant des actions aux règles de réécriture, ou bien lors du parcours de l'arbre de dérivation. Nous choisirons ici la seconde méthode, car **SableCC** ne permet pas d'ajouter des actions aux règles mais il produit l'arbre de dérivation.

La construction de l'arbre abstrait repose sur deux éléments :

- Le visiteur **DepthFirstAdapter** qui réalise un parcours en profondeur de l'arbre de dérivation.
- Le package **sa** (pour syntaxe abstraite) qui définit les classes correspondant aux nœuds de l'arbre abstrait.

### 4. LE VISITEUR **DepthFirstAdapter**

Il est créé automatiquement par **Sablecc**. Il permet de parcourir en profondeur l'arbre de dérivation construit par **SableCC**. Il repose sur deux méthodes : la méthode **apply** qui est définie dans chaque classe **X** correspondant à un nœud de l'arbre de dérivation et les méthodes **public void caseX(X node)** définies dans **DepthFirstAdapter** et qui réalisent le parcours.

Voici l'implémentation de ces deux méthodes pour la classe **APlusExp3**.

- Dans **DepthFirstAdapter.java**

```

public void caseAPlusExp3(APlusExp3 node){
    inAPlusExp3(node);
    if(node.getExp3() != null) node.getExp3().apply(this);
    if(node.getPlus() != null) node.getPlus().apply(this);
    if(node.getExp4() != null) node.getExp4().apply(this);
    outAPlusExp3(node);
}

```
- Dans **APlusExp3.java**

```

public void apply(Switch sw){
    ((Analysis) sw).caseAPlusExp3(this);
}

```

Classe	Interface	implémente	hérite de
	SaNode		
	SaVar SaDec SaExp SaInst		SaNode SaNode SaNode SaNode
SaInst*		SaInst	
SaExp*		SaExp	
SaDecVar SaDecFonc SaDecTab		SaDec SaDec SaDec	
SaVarIndicee SaVarSimple		SaVar SaVar	
SaAppel		SaExp, SaInst	
SaLExp SaLInst SaLDec		SaNode SaNode SaNode	
SaProg		SaNode	

FIGURE 1. Classes et interfaces définies dans le package `sa`. `SaInst*` correspond à toute une série de classes dont le nom a pour préfixe `SaInst`, idem pour `SaExp*`

## 5. LE PACKAGE `sa`

Le package `sa` définit une classe pour chaque type de nœud de l'arbre abstrait. Les différentes classes et interfaces apparaissent dans la figure 1. Le package `sa` est disponible dans le dépôt `git` [https://etulab.univ-amu.fr/nasr/2022\\_compilation](https://etulab.univ-amu.fr/nasr/2022_compilation).

## 6. LA CLASSE `Sc2sa`

C'est la classe que vous devez créer. Elle hérite de la classe `DepthFirstAdapter` et définit la variable d'instance `returnValue` qui permet de mémoriser le dernier nœud de l'arbre abstrait produit. Pour plus de détails, consulter les slides du cours.

```
public class Sc2sa extends DepthFirstAdapter
{
    private SaNode returnValue;
}
```