Overture Technical Report Series
No. TR-002

August 2011

VDM-10 Language Manual

by

Kenneth Lausdah Augusto Ribeiro

Document history

Month	Year	Version
April	2010	

Contents

Chapter 1

Introduction

1.1 Syntax

The syntax is divided into a number of sections:

`Packages` This section allowed the default packages to be defined for `base` and `analysis`:

```
base org.overture.ast.node;
analysis org.overture.ast.analysis;
```

`Tokens` This section defined tokens as tokens coming from a parser which uses the `Token` or external field types either as:

- **Standard Java Types** such as `Integer, Boolean, Long, Char, ...`
- **External classes** where the interface `ExternalNode` is implemented. This provides the AST which the ability to clone the node.
- **External defined Node** where such a node extends `Node` like any other node of the tree. Implementing all analysis and enumerations according to the tree standard.
- **External enum** This is like the standard Java type where the type given is a Java `enum`.

```
bool = 'bool';
java_Integer = 'java:java.lang.Integer';

location = 'java:org.overturetool.vdmj.lex.LexLocation';
LexToken = 'java:node:org.overturetool.vdmj.lex.LexToken';
nameScope = 'java:enum:org.overturetool.vdmj.typechecker.NameScope';
```

`Abstract Syntax Tree` This section describes the actual tree. Names at the top level are considered roots and # are considered sub roots and must be both specified as a root and a child of one or the main roots like `exp` in this case. Fields can be specified using

where the name is giving between the brackets and the type after the colon. A field can also be specified with ( ) which creates a field in the same style but it will not have its parent set when added, this is called a graph field. For the type naming see section **??**.

```
exp {-> package='org.overture.ast.expressions'}
    =   {apply} [root]:exp [args]:exp* [argtypes]:type* (recursive):definition
    |   #Unary
    ;

#Unary {-> package='org.overture.ast.expressions'}
    =   {absolute}
    |   {head}
    |   {mapInverse} (mapType):type.#Map
    ;
```

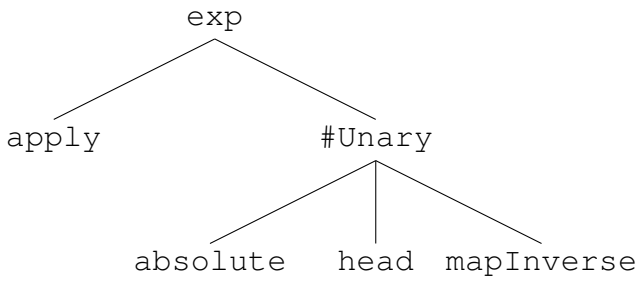

Figure 1.1: AST Example.

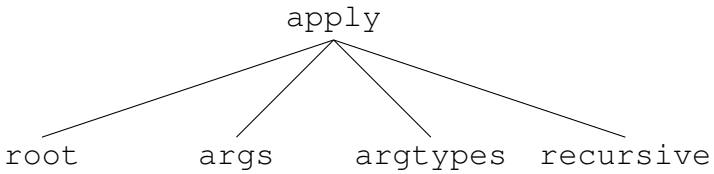

Figure 1.2: Field example for `apply`.

The above example will generate a tree like:

Aspect Declaration hj

```
%exp = [type]:type [location]:location
    ;
```

1.1.1 Hierarchy

1.1.2 Referencing

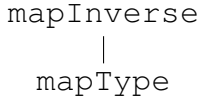

Figure 1.3: Field example for `mapInverse`.

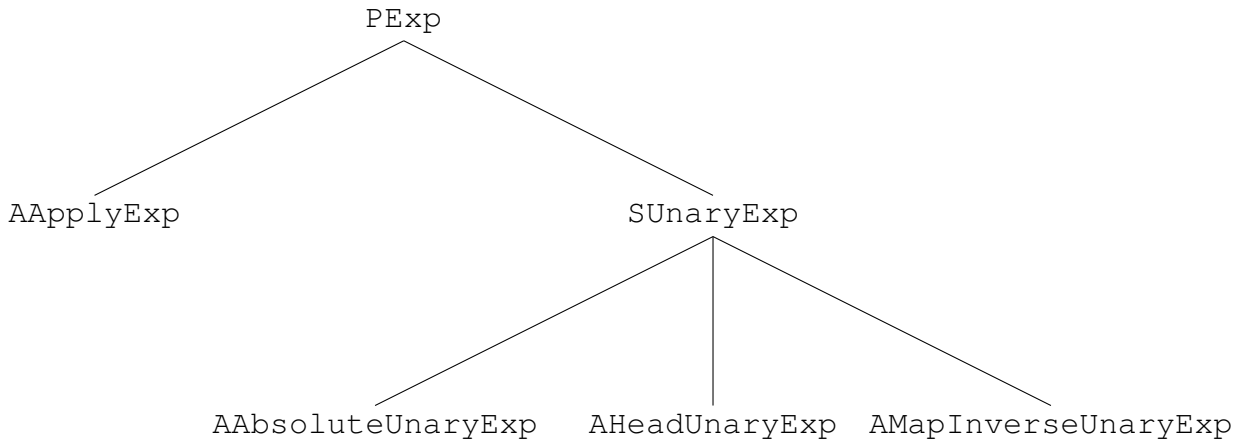

Figure 1.4: AST Example with generated names.