Intermediate Representation of the TassKaf-Frontend

Stephan Diehl, Universität des Saarlandes

May 9, 1997

Contents

1 Introduction

The intermediate representation which is produced by the frontend is a decorated abstract syntax tree. The nodes of this tree are instances of the following C structure:

```
struct NODE
{ int type;
  union { struct NODE *node;
          char* name;
          int value;
        } kids[KIDNUM];
  char* jvm_type;
  char* info;
  int access;
  int offset;
};
```

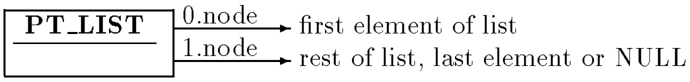
Each node can have several children. These children can be different datastructures: nodes, strings or integers. In other words, so to access the `n`-th kid of a node we have to know what kind of datastructure it contains. Then we can use one of the following schemes.

```
node->kids[n].node
node->kids[n].name
node->kids[n].value
```

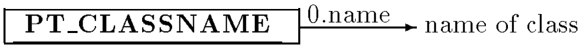
The type of each node is an integer value. In the header files there we defined symbolic constants like `PT_CONST`. In the pictures below the meaning of the components `info`, `jvm_type`,`offset` and `access` are given for each node type.
The file `parstree.c` contains some useful functions to access nodes in a tree. Note, that these functions are already contained in the library `frontend.a`.

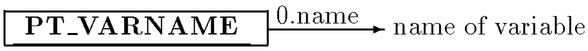
2 Lists

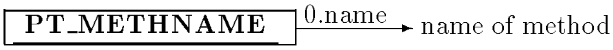
Lists are used for formal parameter lists, argument lists and sequences of statements.

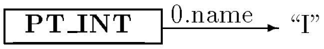


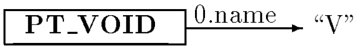
3 Identifier



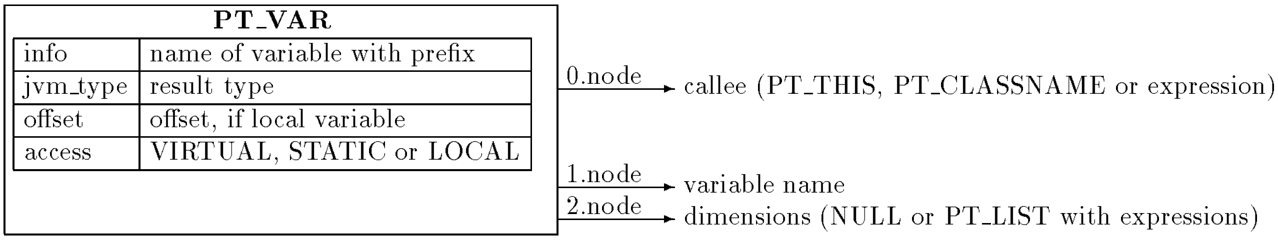


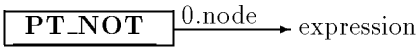


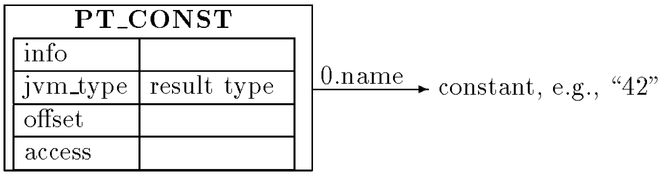


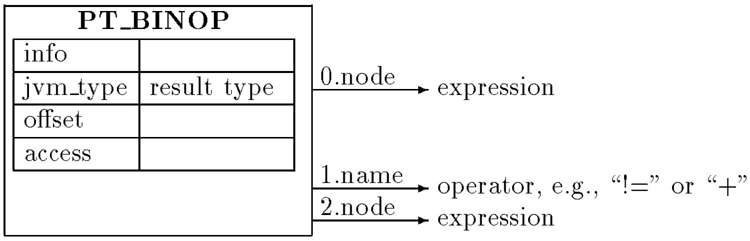


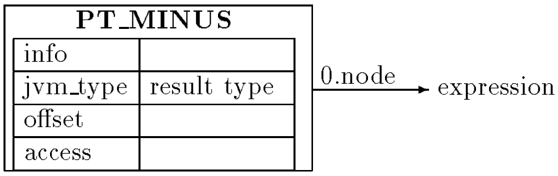
4 Expressions











PT_THIS

info	current class
jvm_type	result type
offset	
access	VIRTUAL

PT_NEWOBJ

info	
jvm_type	result type
offset	
access	

0.name → name of class

PT_NEWARRAY

info	
jvm_type	result type
offset	
access	

0.node → type (PT_CLASSNAME, PT_INT or PT_VOID)

1.node → dimensions, i.e. a list of expressions, its last element can be an expression or PT_RESTDIMS

PT_RESTDIMS

0.value → number of empty dimensions [ ]

PT_METHCALL

info	signature of method
jvm_type	result type
offset	
access	VIRTUAL or STATIC

0.node → callee (PT_THIS, PT_CLASSNAME or expression)

1.node → name of method (PT_METHNAME)

2.node → list of arguments (NULL, expression or PT_LIST with expressions

5 Statements

PT_ASSIGN

0.node → variable (PT_VAR)

1.node → expression

PT_IF

0.node → expression

1.node → statements

2.node → statements (else) or NULL

PT_WHILE

0.node → expression

1.node → statements

PT_DOWHILE

0.node → statements

1.node → expression

PT_FOR

0.node → statement (init)

1.node → expression

2.node → statement (step)

3.node → statements (body)

PT_SWITCH

0.node → expression

1.node → NULL or PT_LIST with cases (PT_CASE)

2.node → statement (default)

PT_BREAK

PT_CASE
- 0.name → constant, e.g. “42”
- 1.node → statements

PT_RETURN
- 0.node → NULL (for void methods) or expression)

6 Declarations

PT_CLASSDEF
- 0.name → name of class
- 1.name → name of superclass
- 2.node → body of class definition (field or method declaration or PT_LIST of those)

In the intermediate representation we use PT_CLASSDEF2 instead. A preprocessing phase generates from the declarations of object and class variables list of assignments. These assignments are used to initialize the class variables when the class is loaded and the object variables when a new instance is created.

PT_CLASSDEF2
- 0.name → name of class
- 1.name → name of superclass
- 2.node → body of class definition (field or method declaration or PT_LIST of those)
- 3.node → list with assignments for body of <init> method
- 4.node → list with assignments for body of <clinit> method

Declarations of class methods (static) are represented by

PT_S_METHDEC

info	
jvm_type	
offset	number of local variables
access	

- 0.node → type (PT_CLASSNAME,PT_INT or PT_VOID)
- 1.node → name of method (PT_METHNAME)
- 2.node → list of parameters (NULL, PT_VARSPEC or PT_LIST with PT_VARSPEC entries)
- 3.node → body of method (variable declaration or statement or PT_LIST of those)

Declarations of object methods are represented by

PT_NS_METHDEC

info	
jvm_type	
offset	number of local variables
access	

- 0.node → type (PT_CLASSNAME,PT_INT or PT_VOID)
- 1.node → name of method (PT_METHNAME)
- 2.node → list of parameters (NULL, PT_VARSPEC or PT_LIST with PT_VARSPEC entries)
- 3.node → body of method (variable declaration or statement or PT_LIST of those)

Declarations of class variables (static) are represented by

PT_S_VARDEC
- 0.node → name of variable and type (PT_VARSPEC)
- 1.node → expression

For object variables we use

PT_NS_VARDEC

- 0.node → name of variable and type (PT_VARSPEC)
- 1.node → expression

For local variables (those declared in the body of a method) we use

PT_LVARDEC

- 0.node → name of variable and type (PT_VARSPEC)
- 1.node → expression

Variable declarations consist of a specification of the type, name and dimensions of a variable (PT_VARSPEC) and an initial value (expression).

PT_VARSPEC

info	type of variable
jvm_type	
offset	offset, if local variable
access	VIRTUAL, STATIC or LOCAL

- 0.node → type (PT_CLASSNAME,PT_INT or PT_VOID)
- 1.node → name of variable (PT_VARNAME)
- 2.value → dimension, if array reference (number of [ ]s)