The JIVE System—Implementation Description

Jörg Meyer, Peter Müller, Arnd Poetzsch-Heffter
Email: [Joerg.Meyer, Peter.Mueller, poetzsch]@Fernuni-Hagen.de
Fachbereich Informatik
FernUniversität Hagen
D-58084 Hagen

Abstract

The Java Interactive Verification Environment JIVE is a verification tool that allows users to prove properties about programs written in a rich subset of sequential Java. This report describes the implementation of JIVE. It gives an overview of the system architecture and provides a detailed description of the different system components.
# Contents

1 Introduction ................................................. 3

2 System Architecture ........................................ 4
   2.1 System Components ..................................... 4

3 Program Information Server ............................... 7
   3.1 Programming and Specification Language ............... 7
   3.2 Architecture ........................................... 7
      3.2.1 Dynamic View ...................................... 8
      3.2.2 Static View ........................................ 9

4 Program Verification Component ......................... 12
   4.1 The Proof Container .................................... 12
      4.1.1 Dynamic View .................................... 12
      4.1.2 Static View ...................................... 13
   4.2 Views .................................................. 17
      4.2.1 Dynamic View .................................... 17
      4.2.2 Static View ...................................... 17

5 Theorem Prover Component ................................ 18
   5.1 Architecture ........................................... 18
      5.1.1 Dynamic View .................................... 18
      5.1.2 Static View ...................................... 21

6 Conclusions .................................................. 24

A Package_jive.PC.Program .................................. 27
   A.1 Interfaces ............................................. 28
      A.1.1 Interface CompRefParserTokenType ............... 28
      A.1.2 Interface CompRefTokenType ..................... 29
   A.2 Classes ................................................ 30
      A.2.1 Class AssignStatement ............................ 30
      A.2.2 Class BlockStatement ............................. 31
      A.2.3 Class CallStatement .............................. 32
      A.2.4 Class CastStatement .............................. 32
      A.2.5 Class CompRef ..................................... 33
      A.2.6 Class EmptyStatement ............................ 34
A.2.7  Class FieldReadStatement ........................................... 35
A.2.8  Class FieldWriteStatement ........................................... 35
A.2.9  Class IfStatement ....................................................... 36
A.2.10 Class CompRefLexer ..................................................... 37
A.2.11 Class InvocationStatement ........................................... 38
A.2.12 Class MethodRef ......................................................... 39
A.2.13 Class ReturnStatement ................................................ 41
A.2.14 Class Statement ........................................................ 41
A.2.15 Class StatementList ...................................................... 43
A.2.16 Class StaticInvocationStatement ..................................... 46
A.2.17 Class UniversalInvocation ............................................ 46
A.2.18 Class VirtualMethod ................................................... 47
A.2.19 Class WhileStatement .................................................. 48
A.2.20 Class CompRefParser .................................................... 49
A.2.21 Class Implementation ................................................... 50

B Package jive ..................................................................... 52
B.1 Classes ..................................................................... 52
B.1.1 Class ExtensionFileFilter ................................................... 52
B.1.2 Class Jive .................................................................. 53

C Package jive.PVC.Container.View ........................................ 55
C.1 Classes ..................................................................... 56
C.1.1 Class AxiomMenuItem ......................................................... 56
C.1.2 Class BackwardMenuItem .................................................... 56
C.1.3 Class FormulaLine ........................................................ 56
C.1.4 Class ForwardMenuItem ...................................................... 57
C.1.5 Class MenuItemLoadException ........................................ 57
C.1.6 Class View ................................................................. 57
C.1.7 Class TreeViewEditor ....................................................... 59
C.1.8 Class OperationMenuItem ................................................... 60
C.1.9 Class PostCondition ......................................................... 60
C.1.10 Class TacticLoadException ........................................... 61
C.1.11 Class TacticMenuItem ....................................................... 61
C.1.12 Class PreCondition ........................................................ 61
C.1.13 Class TreeViewKeyListener ............................................... 62
C.1.14 Class TreeViewRenderer .................................................. 62
C.1.15 Class EmptyLine .......................................................... 63
C.1.16 Class ProgressWindow .................................................... 63
C.1.17 Class TextView ........................................................... 64
C.1.18 Class TreeView ............................................................ 65

D Package jive.PC.PCPeer .................................................... 68
D.1 Classes ..................................................................... 68
D.1.1 Class IllegalCompRefException ........................................... 68
D.1.2 Class Node ................................................................. 68
D.1.3 Class SyntaxException ..................................................... 69
D.14 Class UndeclaredVariableException .................................. 69
D.15 Class PCPeer ............................................................... 70

E Package jive.TPC.TPCPeer ............................................... 77
E.1 Classes ........................................................................ 77
E.1.1 Class TPCException ................................................... 77
E.1.2 Class TPCPeer ............................................................ 77
E.1.3 Class InteractiveTPCPeer ............................................ 79

F Package jive.PVC.tactic .................................................... 80
F.1 Interfaces ...................................................................... 81
F.1.1 Interface Dummy .......................................................... 81
F.2 Classes ........................................................................ 81
F.2.1 Class check_native_call_TACTIC .................................. 81
F.2.2 Class eliminate_assumptions_TACTIC ............................ 81
F.2.3 Class falsify_TACTIC .................................................. 82
F.2.4 Class implementation_block_backward_TACTIC ................. 82
F.2.5 Class simple_weak_backward_TACTIC .......................... 83
F.2.6 Class swb_ptn_TACTIC ................................................. 83
F.2.7 Class swb_TACTIC ....................................................... 84
F.2.8 Class Add_Type_Annotation_TACTIC ............................ 84
F.2.9 Class SeqEqualAssumptForward_TACTIC ....................... 85
F.2.10 Class eliminate_TRUE_TACTIC ................................... 85
F.2.11 Class copy_TACTIC .................................................... 85
F.2.12 Class eliminate_logvar_pre_TACTIC ............................. 86
F.2.13 Class equal_assumptions_forward_TACTIC ................... 86
F.2.14 Class inst_assumpt_ptn_TACTIC ................................ 87
F.2.15 Class unroll_subtype_proof_TACTIC ............................ 87
F.2.16 Class var_forward_TACTIC ........................................ 88
F.2.17 Class replay_TACTIC ................................................ 88

G Package jive.PVC.Container.Formula .................................. 89
G.1 Interfaces ...................................................................... 90
G.1.1 Interface PVSTokenTypes ............................................ 90
G.1.2 Interface FormulaParserTokenTypes ............................. 94
G.1.3 Interface SubstWalkerTokenTypes ................................ 99
G.1.4 Interface UnparseWalkerTokenTypes ............................ 104
G.1.5 Interface CollectVarWalkerTokenTypes ......................... 109
G.2 Classes ....................................................................... 114
G.2.1 Class BooleanFormula ................................................ 114
G.2.2 Class FormulaSyntaxException ................................... 119
G.2.3 Class HashMultiSet .................................................... 119
G.2.4 Class FormulaLexer .................................................... 120
G.2.5 Class FormulaParser .................................................. 123
G.2.6 Class SubstWalker ..................................................... 125
G.2.7 Class Formula ........................................................... 127
G.2.8 Class UnparseWalker .................................................. 131
H Package jive.PVC.Container

H.1 Classes

H.1.1 Class Assumptions

H.1.2 Class CompRefString

H.1.3 Class FormulaString

H.1.4 Class LogicalVarString

H.1.5 Class Rule

H.1.6 Class Tactic

H.1.7 Class ProgVarString

H.1.8 Class subst_backward

H.1.9 Class call_backward

H.1.10 Class ReqException

H.1.11 Class all_backward

H.1.12 Class all_forward

H.1.13 Class assumpt_elim_backward

H.1.14 Class assumpt_elim_forward

H.1.15 Class ContextInfo

H.1.16 Class assumpt_intro_backward

H.1.17 Class assumpt_intro_forward

H.1.18 Class block_backward

H.1.19 Class block_forward

H.1.20 Class call_forward

H.1.21 Class var_backward

H.1.22 Class Triple

H.1.23 Class check_assign_axiom

H.1.24 Class check_assumpt_axiom

H.1.25 Class check_false_axiom

H.1.26 Class check_field_read_axiom

H.1.27 Class check_field_write_axiom

H.1.28 Class check_return_axiom

H.1.29 Class class_backward

H.1.30 Class class_forward

H.1.31 Class conjunct_backward

H.1.32 Class conjunct_forward

H.1.33 Class disjunct_backward

H.1.34 Class disjunct_forward

H.1.35 Class ex_backward

H.1.36 Class ex_forward

H.1.37 Class if_backward

H.1.38 Class if_forward

H.1.39 Class implementation_backward

H.1.40 Class implementation_forward

H.1.41 Class inst_assign_axiom

H.1.42 Class inst_assumpt_axiom

H.1.43 Class inst_false_axiom
H.1.44 Class inst_field_read_axiom .......................... 158
H.1.45 Class inst_field_write_axiom .......................... 158
H.1.46 Class inst_return_axiom ................................. 158
H.1.47 Class inv_backward ................................... 159
H.1.48 Class inv_forward ....................................... 159
H.1.49 Class var_forward ....................................... 160
H.1.50 Class invocation_backward ............................ 160
H.1.51 Class invocation_forward .............................. 161
H.1.52 Class seq_back_backward .................................. 161
H.1.53 Class seq_front_backward .................................. 161
H.1.54 Class seq_forward ....................................... 162
H.1.55 Class static_invocation_backward ...................... 162
H.1.56 Class static_invocation_forward ....................... 163
H.1.57 Class strength_backward .................................. 163
H.1.58 Class check_cast_axiom .................................. 164
H.1.59 Class check_empty_axiom .................................. 164
H.1.60 Class subst_forward ..................................... 164
H.1.61 Class strength_forward .................................. 165
H.1.62 Class subtype_backward ................................... 165
H.1.63 Class subtype_forward ................................... 166
H.1.64 Class weak_backward ..................................... 166
H.1.65 Class weak_forward ....................................... 167
H.1.66 Class while_backward .................................... 167
H.1.67 Class while_forward ...................................... 167
H.1.68 Class Sequent ............................................. 168
H.1.69 Class inst_cast_axiom .................................... 170
H.1.70 Class inst_empty_axiom ................................... 170
H.1.71 Class ProofTreeNode ....................................... 171
H.1.72 Class ProofContainer ............................... 176
Chapter 1

Introduction

Within the research project Lopez, we deal with the construction of so-called logic-based programming environments [PH97b]. These are software tools that support the formal specification and verification of programs. The JIVE system is a prototype of such a programming environment. It allows users to verify properties of specified programs. Input programs are written in a sequential Java subset called SVENJA [MMPH97]. We use a interface specification language that is based on the two-tiered specification approach developed in the Larch project [PH97b, MPH97, GH93]. For verification, we apply a Hoare-style programming logic [PHM98, PHM99].

Verification tools are very complex pieces of software [GMP90, MPH00a]. They combine a compiler frontend (for scanning, parsing, and the static analysis of specified programs), an interactive program prover component with graphical user interface, proof strategies [MPH00b], and elaborate user guidance, and a theorem prover to exploit state-of-the-art technology for reasoning in program-independent logics. As a result of this complex structure, our prototype is written in a number of different programming languages (such as Java, C, Lisp, etc.), uses various development tools (for instance lex, yacc, MAX [PH97a], ANTLR [ANT00]), and integrates a complete theorem prover (PVS [COR+95]). This report gives an overview of the system architecture and provides a detailed description of the different system components. For each component, we describe:

- the dynamic view, which gives insight to the component structure and behavior (in particular communication) at runtime.
- the static view, which explains the structure of the implementation and highlights some interesting coding techniques.

This report is not completely self-contained. It is supposed to be read in combination with the commented code. Its purpose is to enable readers to become familiar with the JIVE implementation in order to make changes, fix bugs, and enhance the functionality.

Overview. We provide an overview over the architecture of the JIVE system in Chapter 2 and explain the main system components, namely the program information server, the program verification component, and the theorem prover component in Chapters 3, 4, and 5. Our conclusions and directions for further work can be found in Chapter 6. The appendix contains an automatically generated documentation of the implementation parts written in Java. It provides further insight into the design of the various classes.
Chapter 2

System Architecture

Within this section we give an overview of the overall system architecture and describe 1. how the system is build of components and 2. how the components are connected.

2.1 System Components

To give an impression of how the components interact, we describe in the following what kind of input is used to set up a proof session and what kind of data is exchanged between the system components. All mentioned System parts and data are shown in Figure 2.

Components and Data. Each proof projects starts with The architecture is based on five components: 1.) The syntax analysis component that reads in and analyzes annotated programs and generates the program proof obligations. 2.) The program information server that makes the static program information gathered in the analysis phase available to other parts of the system. 3.) The program prover component managing the program proofs. 4.) Views to visualize program proofs and to control proof construction. 5.) The theorem prover to solve program independent proof obligations. In our current implementation, we use PVS for general theorem proving.

The program proof component encapsulates the construction of program proofs. It provides two things: (1.) A container which stores all information about program proofs and (2.) an interface which provides operations to create and modify proofs within this container. Since the content of the proof container represents the program proof state, it is strongly encapsulated to the rest of the system. Modifications of the proof state can only be achieved by operations of the container interface. Therefore correctness of proofs is ensured by the correctness of the basic container operations.

During program proof construction, various information about the underlying program is needed by the program proof component: The structure of the abstract syntax tree, results of binding and type analysis, and the program unparsing for visualization. This kind of information is provided by the program information server. In contrast to a compiler frontend, all information computed during static program analysis has to be available online after the analysis.

The verification of a program is based on three formal texts: 1.) The PVS prelude containing two parts: (a) the formalization of the object-store; (b) the specification of abstract data types used in program annotations. Whereas the former part is program independent,
the latter may be program dependent. 2) The Anja prelude containing the specifications of predefined and library classes and interfaces. 3) An Anja program, i.e. a program in our Java subset together with a suitable interface specification. Annotations are formulated in a language based on the specification language of the underlying theorem prover, i.e. PVS in our case. They may refer to program variables and use abstract data types specified in the PVS prelude.

Syntax and static semantics of the Anja-program is checked with a compiler frontend which also performs the syntax analysis of the annotations. Annotations are formulated in a language based on the specification language of the underlying theorem prover (here PVS). They may contain program variables and use abstract data types specified in theories of the theorem prover. To check a specification formula \( \mathcal{E} \), \( \mathcal{E} \) is sent to the theorem prover which checks, whether the expression is correctly typed w.r.t. the used abstract data types and the formalization of the object store.

From the described sources, the syntax analysis component generates three things: 1. The program proof obligations which need to be proven to guarantee that the program fulfills its specification. They are entered into the proof container. 2. Program dependent theories formalizing some of the declaration information of the program for the theorem prover. 3. The abstract syntax tree decorated with information of the static analysis. It is managed by the program information server.

After syntax and static analysis, the system is set up for interactive proof construction. The user constructs program proofs using basic proof operations and tactics. The views and controllers provide access to the proof state. Program independent proof obligation are verified with the general theorem prover. The program prover monitors the overall proof process and signals the completion of proof tasks.
Interfaces and Proof State. Because we used for each task tools which are quite sophisticated in its area, the JIVE has a heterogeneous architecture. The Program Verification Component and the views are implemented in Java. The program information server and the syntactical analysis of ANJA programs is implemented in C and the PVS Theorem Prover Component is implemented using a Lisp dialect and accessible via Emacs using its Emacs Lisp interface. The communication among these different components is enabled by different kinds of interfaces. The following enumeration gives an overview over the used interfaces. More details are described at the subsections for the different system components:

- Text file interface. Text files are generated and reused during system use for theories containing formal PVS descriptions, abstract data types used within specifications and program proofs etc. Some of the reused theories are generic with respect to class types, interface types, and attribute identifiers used in the ANJA programs. Furthermore proof obligations, which need to be proved interactively with the PVS System are generated in text files and loaded by PVS.

- Java method call interface. Java method calls are used wherever system components are implemented in Java and run within one instance of the Java Virtual Machine, e.g. the proof container and the views.

- Java native interface. Java native method calls are used to communicate between the system components implemented in C and the Java implemented parts. This includes the syntactical analysis of ANJA programs and the program information server.

- Unix Socket interface. The PVS System is attached to JIVE via a socket communication, because it run in a separate process.

- Emacs lisp function call. Within PVS, PVS is controlled by an Emacs process, which provides communication facilities by a Emacs functions. Thus we implemented the interface for PVS in Emacs lisp by providing a collection of Emacs functions.

The system state of JIVE is (mostly) not distributed over all components and only program verification component holds the system state. There is one exception, where the system state is distributed: If the user proves generated theorems interactively within the TPC component, the result of this work is not send to the program verification component online. After finishing proving within the TPC component the user initiates a transfer of proof results to the program verification component. Simultaneous work in both components is not allowed, thus it is not possible to get into an inconsistent state this way.
Chapter 3

Program Information Server

The program information server PIS is responsible for all actions that directly deal with the input program and its interface specification:

- It performs scanning, parsing, and static analysis;
- It generates the program-dependent theories for the universal specification (see Subsection 5.1);
- It provides the other system components with information about the program and its interface specification (such as type information, variable and method declarations, pre- and post-conditions, etc.)

In the following, we sketch the programming language SVENJA and the interface specification language ANJA used in the JIVE system, and describe the architecture of the PIS component.

3.1 Programming and Specification Language

The programming language SVENJA used in the JIVE system is a rich subset of Java [GJS96]. It provides classes and interfaces, subtyping and inheritance, instance fields, instance and static methods, static and dynamic method binding, encapsulation, a variety of statements, and primitive expressions.

The interface specification language (ISL) ANJA is based on the ISL developed in [PH97b]. It features requires-clauses, pre- and postconditions for method specifications, and class invariants. The formulas occurring in interface specifications are based on the formulas of the PVS language [OSR93]. They are essentially multisorted first-order formulas.

A detailed description of SVENJA and ANJA (including the formal syntax and an example) can be found in [MMPH97].

3.2 Architecture

In this subsection, we describe the dynamic and static view of PIS' architecture.
3.2.1 Dynamic View

The dynamic view of an architecture comprises the runtime component structure and the communication with other system components. Both aspects of the PIS component are described in the following.

The dynamic structure of PIS is very simple. For the other JIVE components, it acts as a homogeneous component the interface of which consists of a set of static methods and static fields. PIS does not have a graphical user interface. Besides several data structures to store information about the program and its specification, PIS does not consist of any objects.

Internally, PIS consists of two components: The frontend analyzes the input program and specification\(^1\), and a so-called peer is used to interact with other JIVE components. That is, the peer serves as a broker between other JIVE components and the frontend; all queries to the frontend must go through the peer. The frontend is implemented with the MAX tool [PH97a], the peer is written in Java. They communicate via the Java Native Interface [Sun98].

The Checker. In its current version, MAX can create and hold only one abstract syntax tree for each program run. Therefore, it is not possible to close a project and open a new one in one JIVE session. Instead, the JIVE process has to be ended and restarted.

To avoid restarts when the input program or its specification are syntactically incorrect, the PIS component provides a so-called checker. The checker's functionality is a subset of the frontend's functionality: It scans, parses, and checks the input program and specification. In contrast to the frontend, the checker is not permanently attached to the JIVE system. Instead, it can be run as a separate process to check input programs. This way, the input can be checked and—in case that errors are detected—they can be fixed without having to restart JIVE. The checker can be run by invoking a method of the peer. It communicates with the peer via streams (i.e., the standard out streams of the checker process is used to inform the peer whether the checked program contains errors).

Component References. Internally, PIS stores the abstract syntax tree in the MAX format. To provide an abstract view to the AST, PIS uses so-called component references to represent positions in the abstract syntax tree. To speed up accessing the AST, PIS stores the component references of all methods (method implementations and virtual methods) in a hash table. Component references allow other JIVE components to address positions in the AST, to extract information, and to navigate in the tree. Component references are also used in the representation of Hoare triples.

For some operations, users of the JIVE system have to textually specify component references. There are four forms of textual representations:

\[ T:m(T_1,\ldots,T_n) \] specifies the virtual method for method \( m \) in class \( T \) with parameter types \( T_1,\ldots,T_n \).

\[ T0m(T_1,\ldots,T_n) \] specifies the method implementation for method \( m \) in class \( T \) with parameter types \( T_1,\ldots,T_n \).

---

\(^1\) Since the syntax and typing of formulas is based on the PVS language (with its undecidable type system) and relies on PVS theories, PIS cannot type check formulas in interface specifications. Instead, formulas are type checked by the TPC before they are used by any JIVE component (see Section 5).
\( T_{\text{m}}(T_1, \ldots, T_n)[i] \) specifies statement \( i \) in the body of the given method implementation. The numbering of statements in a method body is not specified; users can obtain the number from the program view of the user interface.

\( T_{\text{m}}(T_1, \ldots, T_n)[i,j] \) specifies the statement list from statement \( i \) to statement \( j \) in the body of the given method implementation.

To parse such textual representations, PIS temporarily creates a scanner and parser for component references, processes the input, and creates an appropriate object.

**The PIS Interface.** PIS acts as a server for other JIVE components. That is, the PVC and the central control can direct queries in the form of method invocations to PIS’ peer. However, since the peer is realized as a set of static methods, virtually any JIVE component could access PIS’ interface. The interface of PIS provides three groups of methods:

1. Initialization methods that are called at the beginning of a JIVE session (e.g., to analyze the input program, to generate program-dependent theories, and to generate the proof obligations stemming from the interface specification). Initialization methods are called by the central control and the PVC.

2. A method to parse textual representations of component references based on an ANTLR parser [ANT00]. This method is called by the PVC.

3. Methods to extract information about the program and the interface specification. These methods are the main part of the PIS interface. They are called by the PVC. The selection of methods is rather ad hoc, based on the needs of the PVC and the logic implemented there. Different axioms and rules will require to extend the PIS interface.

### 3.2.2 Static View

In the following, we describe the implementation of the PIS component. The code for PIS resides in the directory jive/PC. It consists of three major parts: The peer, the frontend, and the implementation of component references (the code of the checker is a subset of the frontend’s implementation).

**The Peer.** The peer is implemented in package jive.PC.PCPeer. The package contains the following classes:
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCPeer</td>
<td>contains the static methods described above. Many of the methods are native methods. These methods are implemented in the frontend (see below).</td>
</tr>
<tr>
<td>Node</td>
<td>is used to store occurrences in the MAX abstract syntax tree on the Java side. Nodes contain a sort index and an element index that specify a unique position in the AST. Thus, the hash table (field PCPeer.comrefs) maps component references to objects of class Node.</td>
</tr>
<tr>
<td>IllegalCompRefException</td>
<td>is thrown when an illegal textual representation for a component reference is parsed.</td>
</tr>
<tr>
<td>SyntaxException</td>
<td>indicates that the input program contains errors or that a method tries to get information about a class, variable, etc. of the input program that does not exist.</td>
</tr>
</tbody>
</table>

**The Frontend and the Checker.** The frontend and the checker are generated by lex, yacc, and MAX. The code is located in the directory jive/PC/FrontEnd. It consists of the following units (an ‘f’ or a ‘c’ in the first column indicates that the unit is used by the frontend resp. the checker only; all other units are part of the frontend and of the checker):

<table>
<thead>
<tr>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pis.l</td>
<td>contains the scanner specification in lex format.</td>
</tr>
<tr>
<td>pis.y</td>
<td>contains the parser specification in yacc format.</td>
</tr>
<tr>
<td>yyerror.c</td>
<td>supplements the parser specification.</td>
</tr>
<tr>
<td>pis.m</td>
<td>is the MAX specification of the abstract syntax and the static analysis. Furthermore, it contains functions for the generation of program-dependent theories, for the un parsing of program parts, and for answering queries from the PVC.</td>
</tr>
<tr>
<td>max_help.c</td>
<td>supplements pis.m. It contains auxiliary functions for the MAX specification.</td>
</tr>
<tr>
<td>addInMethodHdrList.c</td>
<td>supplements pis.m. It contains auxiliary functions for the MAX specification.</td>
</tr>
<tr>
<td>checkMethodheaderList.c</td>
<td>supplements pis.m. It contains auxiliary functions for the MAX specification.</td>
</tr>
<tr>
<td>f pis help.c</td>
<td>contains the implementation of most of PCPeer’s native methods.</td>
</tr>
<tr>
<td>f getDirectSubtypes.c</td>
<td>implementation of the peer’s native method</td>
</tr>
<tr>
<td>f isDirectSubtype.c</td>
<td>implementation of the peer’s native method</td>
</tr>
<tr>
<td>f isSubtype.c</td>
<td>implementation of the peer’s native method</td>
</tr>
<tr>
<td>f getMethod.c</td>
<td>implementation of the peer’s native method</td>
</tr>
<tr>
<td>f insertCompRefs.c</td>
<td>implementation of the peer’s native method</td>
</tr>
<tr>
<td>f insertSequentsNative.c</td>
<td>implementation of the peer’s native method</td>
</tr>
<tr>
<td>f isTypeName.c</td>
<td>implementation of the peer’s native method</td>
</tr>
<tr>
<td>f axioms.c</td>
<td>contains functionality for the generation of program-dependent PVS theories.</td>
</tr>
<tr>
<td>addit.c</td>
<td>contains functionality to add predefined create-methods to each SVENJA class.</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inChecker.c</td>
<td>contains just a global variable that determines whether the frontend or the checker is run.</td>
</tr>
<tr>
<td>getTheoryName.c</td>
<td>contains functionality to retrieve the name of the file that contains the program-dependent theories.</td>
</tr>
<tr>
<td>getBaseName.c</td>
<td>contains functionality to retrieve the base name of the project.</td>
</tr>
<tr>
<td>checker.c</td>
<td>contains the main function of the checker.</td>
</tr>
<tr>
<td>prelude.anja</td>
<td>contains the predefined types that are part of each SVENJA program (see [MMPH97]). When the frontend reads a SVENJA file, it adds the predefined types to the AST.</td>
</tr>
</tbody>
</table>

To integrate the frontend with the other JIVE components, we generate a dynamic library from the object files. This library is loaded by the central control when JIVE is started. See makefiles and the JNI documentation [Sun98] for details. The checker is compiled and linked to an executable program that is invoked by the peer via Java's Runtime-class.

**Component References.** Component references are implemented in packagejive.PC.Program. This package contains two logical units: The classes to represent positions in the AST (i.e., the actual component references) and the scanner and parser for textual representations for component references, which are used to analyze user input.

All positions in the AST are modeled by subclasses of class CompRef. However, CompRef objects do not form a tree. They just represent positions in the MAX AST. Each position is modeled by a method (specified by the name of the method, the surrounding class, and the parameter types). For statements and statement lists, the position additionally contains one resp. two statement numbers. The statements of the input program are numbered when the AST is built. The numbering is not specified.

According to the COMP-parts of our programming logic, there are classes for the various kinds of statements (which are all subclasses of Statement) and classes for virtual methods and method implementations (subclasses of MethodRef). The different classes provide methods to extract information (e.g., the types of formal method parameters) and to navigate in the AST (e.g., to get the body of a while-loop). The functionality provided by the CompRef subclasses was determined by the needs of our current programming logic. Different logics or more elaborate proof strategies might require additional methods.

Whenever information about the AST is retrieved, the according CompRef is used to look up a Node object in the compref hash table (see above). This Node object contains the occurrence of the method declaration in the AST. For statements and statement lists, the AST can be traversed from this occurrence to find the statement occurrence(s). This functionality is crucial for most implementations of PCPeer's native methods. It is implemented by the following functions in file pisehlp.c: getOccurrence, getStatementOccurrence, getMethodNode, and searchStatementNode.
Chapter 4

Program Verification Component

The implementation of the Program Verification component can be split up into the proof container part and the view part. The proof container part provides all functionality to construct proofs and the view part provides the user interfaces for the program verification component. I.e. it provides visual representations of proof data structures and and possibilities to control proof construction.

4.1 The Proof Container

The dynamic view of the proof container describes, how program proof obligations are inserted into the proof container at system startup and how program proof are constructed within it using proof operations and strategies.

4.1.1 Dynamic View

In this subsection we describe the most important methods of the proof container implementation. At system startup one proof container instance is created. Beneath the initialization of all auxiliary data structures, the proof container loads all proof operations and strategies, which are provided by separate Java files and makes them available for use. This technique enables a modular development of proof operations.

When the proof container data structure is set up, it notifies the PCPeer component to send all program proof obligations generated out of the program specification using the newProject-method (see Fig. 4.1). The PCPeer uses the proof containers insertGoal-method to send the program proof obligations to the proof container. The system is now ready to construct program proofs. In every system state the container contains a collection of proof trees which directly represent the proof trees of the Hoare logic proofs, which can be extended to program proofs for the generated proof obligations using the following proof operations:

- Using Proof operations provided by Proof Operations.
- Using Using control operations like split, concat, remove root, remove subtree, remove tree with the obvious functionality. The operations have to be provided separately, because the underlying programming logic does not provide this.
- Using strategies which combine sequences of the above described operations.
4.1.2 Static View

In the following we describe the main Java classes, which are used to implement the proof container. Furthermore we describe the class patterns which have to be used to implement proof operations and strategies. For security reasons all implementation parts are encapsulated using Java language features in such a way that is not possible to invalidate the state of the proof container from implementation parts which do not belong directly to the proof container implementation. This entails, that proof operations and control operations belong to the container and strategies do not. Thus (1) strategies can be provided by system users later in a secure way and (2) strategies could be recorded during the system use and the recorded proof operation sequence could be provided as tactic. The last feature is not supported in the current state.

A Data Structure for Program Proofs. The following classes build the collection of main classes which is used to implement the proof container. Prooftrees are represented by object structures, which itself directly represent all parts of which Hoare-logic Proof consists, like sequents triples assumptions, etc.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProofContainer</td>
<td>The main class representing a proof container. A ProofContainer object contains a collection of ProofTreeNode objects which represent the roots of prooftrees. Furthermore contains a HashTable to manage logical variables used within proofs.</td>
</tr>
<tr>
<td>ProofTreeNode</td>
<td>Prooftrees consist of ProofTreeNode-Objects. Each ProofTreeNode represents a Node within a ProofTree and contains a Sequent object for the represented Sequent.</td>
</tr>
<tr>
<td>Sequent</td>
<td>Sequents-objects represent sequents from the underlying programming logic and contain an Assumptions-object which represents the assumptions and a Triple object, which represents the sequents triple.</td>
</tr>
<tr>
<td>Triple</td>
<td>A Triple-object represents a Sequent without assumptions and can only be contained within a sequent object, thus a triple is not a subtype of Sequent. Sequents with an empty assumption set are represented by a Sequent object with an empty assumption set. A triple consists of two BooleanFormula Objects, which represent the pre- and postcondition of a triple and a CompRef object, which refers to the program part of the triple.</td>
</tr>
<tr>
<td>Assumptions</td>
<td>An Assumptions represents a set of triples. Within the current implementation and according to the underlying programming logic, only Triples with referring to virtual methods and method implementations are allowed.</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BooleanFormula</td>
<td>A BooleanFormula object represents a boolean formula in the syntax of the PVS language (see below).</td>
</tr>
<tr>
<td>CompRef</td>
<td>A CompRef object refers a program part within the underlying programs like method and statements (see 3.2.2).</td>
</tr>
</tbody>
</table>

**Proof Operations.** As described above, proof operations are not directly implemented within the proof container, but dynamically loaded during the initialization of a ProofContainer object. To provide this feature, proof operations have implement according to the patterns described in the following. We provide three kinds of patterns for 1. Forward proof operations, 2. backward proof operations and 3. axioms. These three patterns are basically identical up to different naming conditions to distinguish the kind of operation. Furthermore each proof operation has to be a subtype of class Rule and have to member of the proof container package jive.PVC.Container.

Each proof operation has to use the following naming condition, where op is the name of the operation:
```java
package jive.PVC.Container;  // implementation

public class op_forward extends Rule
{
    // implementation

    public op_forward(ProofContainer c, View v) {
        super(c, v);
    }
}
```

Each proof operation has to contain a constructor of the following style:

```java
public class op_forward(ProofContainer c, View v) {
    super(c, v);
}
```

The correctness of each proof operation is preserved by the correctness of the implementation of the proof operation, thus implementation a proof operation has to be done very accurately. The functionality of a proof operation is placed within a method with the name call:

```java
ProofTreeNode call(p1, ..., pn) throws ReqException
```

This method always has to return a ProofTreeNode, in case of a forward operation the new prooftree root and in case of a backward operation one of the new goals (or closed proof tree nodes in case of an axiom use). The parameters to this method are one ProofTreeNode and some other Strings describing logical variables, program variables, or program references. In case of a forward operation there may be two parameters of type prooftree node to combine two prooftrees to one proof. A proof operation first has to check all demands which the parameters have to meet and throw a ReqException, if some are not met. The getBackwardParameters() method has to return an array of objects, which contains the parameters needed to use this rule in backward direction without the ProofTreeNode parameter. These parameters are needed to replay proofs backward. The following implementation of the conjunct_forward operation gives an example for a proof operation implementation: First, three requirements
to the parameters are checked: 1. Both actual parameters are roots of prooftrees, 2. the
assumption sets of both sequents of the prooftree roots do not differ, and 3. the sequent for
the new root is created and both prooftrees given as parameters are combined in one prooftree
with the new sequent as a root.

package jive.PVC.Container;
import jive.PVC.Container.View.*;
import jive.PC.Program.*;
import jive.PVC.Container.Formula.*;

public class conjunct_forward extends Rule {

    public conjunct_forward(ProofContainer c, View v) {
        super(c,v);
    }

    ProofTreeNode call(ProofTreeNode pta, ProofTreeNode ptb) throws ReqException {
        pta.checkProofTreeRoot();
        ptb.checkProofTreeRoot();

        Sequent sa = pta.getSequent();
        Sequent sb = ptb.getSequent();

        // the assumption sets of a and b must be identical
        sa.getAssumptions().checkAssumptionsEqual(sb.getAssumptions());

        // the compref of ptas root equals the compref of ptbs root
        sa.getCompRef().checkCompRefEqual(sb.getCompRef(), "");

        //construct the enriched prooftree
        Sequent newroot = new Sequent(sa.getAssumptions(),sa.getPre().and(sb.getPre()),
                                      sa.getCompRef(),sa.getPost().and(sb.getPost()));

        return ProofTreeNode.insertRoot(newroot, pta, ptb);
    }

    protected Object[] getBackwardParameters() {
        return Rule.noparameters;
    }
}

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>The supertype for all proof operation. All proof operations have to be a subtype of Rule.</td>
</tr>
<tr>
<td>CompRefString</td>
<td>A String representing a CompRef</td>
</tr>
<tr>
<td>ProgVarString</td>
<td>A String representing a program variable</td>
</tr>
<tr>
<td>LogicalVarString</td>
<td>A String representing a logical variable</td>
</tr>
<tr>
<td>FormulaString</td>
<td>A string representing a formula</td>
</tr>
<tr>
<td>ReqException</td>
<td>A requirement exception is thrown, if a demand to a parameter is not met.</td>
</tr>
</tbody>
</table>
**Strategies.** Strategies are implemented similar to proof operations. There are four major differences between strategies and proof operations: 1. Strategies reside in the package jive.PVC.Tactic, thus strategies cannot invalidate the proof container state. 2. the name pattern differs, 3. they have to be subtypes of the class Tactic\(^1\) from the ProofContainer package, and 4. strategies do not need to implement the getBackwardParameters method. The implementation of a strategy reuses other proof operations and strategies. To do this, they have to create an appropriate object of the used operation or strategy and use its call method. Strategies do not have to implement the getBackwardParameters, because proof construction is ultimately based on the use of proof operations, therefore within a proof replay only proof operations are used, but never strategies.

```java
package jive.PVC.Tactic;

class op_TACTIC extends Rule {
    // implementation
}
```

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tactic</td>
<td>The superclass</td>
</tr>
</tbody>
</table>

The following example show a simple tactic, which uses the assumpt_intro_backward proof operation, to remove all assumptions from an open goal within a proof container.

```java
package jive.PVC.tactic;
import jive.PVC.Container.*;
import java.util.*;
import jive.PVC.Container.*;
import jive.PVC.Container.View.*;
import jive.PVC.Container.Formula.*;
import jive.PC.PCPeer.*;
import jive.PC.Program.*;

public class eliminate_assumptions_TACTIC extends Tactic {

    private assumpt_intro_backward aib;
    public eliminate_assumptions_TACTIC(ProofContainer c, View v) {
        super(c,v);
        aib = new assumpt_intro_backward(c,v);
    }

    public ProofTreeNode call(ProofTreeNode ptn) throws ReqException {
        container.dispatchMessage("eliminating assumptions for " + ptn.getId());

        Enumeration e = ptn.getAssumptions().elements();
        while(e.hasMoreElements()) {
            Triple a = (Triple)e.nextElement();
            ptn = aib.call(ptn,new FormulaString(a.getPre().toString()),
                a.getCompRef().getChString(),
                new FormulaString(a.getPost().toString()));
        }
    }

\(^1\)For historical reasons the superclass for tactics is called Tactic instead of Strategy.
```
return ptn;
}
}

For historical reasons the superclass for strategies is called Tactic instead of Strategy.

**Formulas.** Formulas needed within the proof container are provided by

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula</td>
<td>Represents arbitrary formulas</td>
</tr>
<tr>
<td>BooleanFormula</td>
<td>Represents boolean valued formulas used e.g. for pre- and postconditions in triples.</td>
</tr>
</tbody>
</table>

4.2 Views

4.2.1 Dynamic View

4.2.2 Static View

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
</table>
Chapter 5

Theorem Prover Component

The theorem prover component TPC integrates the general-purpose proof checker PVS [COR+95] into the JIVE system. In its current version, the TPC is used to type check formulas and user-defined theories as well as to interactively prove lemmas and to send the proof status to the JIVE system.

5.1 Architecture

The architecture of the TPC resembles the structure of PIS. Both components integrate a component that is not implemented in Java via a peer. We describe the architecture of the TPC in the following.

5.1.1 Dynamic View

Since PVS has to run as separate interactive process, integrating PVS into JIVE is somewhat difficult. In particular, the user can interactively bring PVS in a state that is not expected by the JIVE system. To keep these problems as small as possible, we decided to use PVS in batch mode for all tasks that do not require user interaction (such as type checking), and an additional PVS process for interactive tasks (theorem proving). Both PVS processes are attached to the JIVE system via a peer. We describe the structure and communication of both parts of the TPC in the following.

PVS in Batch Mode. The batch part of the TPC (batch TPC for short) consists of (1) the PVS component, (2) the peer that allows JIVE to communicate with the PVS process, and (3) a so-called output listener that receives the process output of the PVS process (e.g., error messages) and sends them to JIVE’s central control for displaying. The PVS component in turn consists of the PVS process and an Emacs process that serves as a platform for PVS. Fig. 5.1 shows the structure of the batch TPC.

Communication. The batch TPC peer communicates with other JIVE components via method invocations. It provides methods to start and quit sessions, to open new projects, and to type check formulas.

The batch TPC peer and the Emacs process communicate via socket connection. The protocol is very simple: All actions are initiated by the peer. To send a command to PVS,
the peer creates a message object, which is then marshaled and sent to Emacs, which unmarshals the message, extracts the command, and invokes the appropriate LISP function. After executing the command, Emacs sends a reply (i.e., an acknowledgment or an error message) back to the batch TPC peer. The communication is synchronous. That is, after sending a message, the peer blocks until it receives a reply. The commands are described together with the message format in Subsection 5.1.2.

The output listener receives the process output (that is normally sent to standard out) from the Emacs process and passes it to JIVE’s central control. It is an independent thread that loops infinitely until JIVE is shut down.

**System Startup.** When the JIVE system is started, the central control invokes an initialization method of the TPC peer. Initiated by this method, the following actions take place:

1. The peer creates a server socket for the connection to PVS;
2. It starts PVS, passing the host and port of the server socket and a LISP file that is executed by Emacs;
3. It creates and starts an output listener;
4. Emacs connects to the server socket; the connection is then established.

**System Shutdown.** To shut down the batch TPC, a method of the peer is invoked. This method sends a shutdown command to the Emacs process and closes the socket connection.

**Opening Projects.** When a new project is opened, the central control invokes a method of the TPC peer after PVS has generated the program-dependent theories for the new project and before the proof obligations are inserted into the proof container (this is to ensure that there is a valid context to type check the formulas in the sequents). TPC then changes the PVS context to the base directory of the new project and type checks the theory context (i.e., the user-defined and the generated theories, see below).

**Interactive PVS.** The interactive TPC component consists of a peer and the PVS and Emacs processes. It is only temporary attached to a running JIVE system. Upon user request, all lemmas of a proof session are written to a theory. Then, a peer object is created and a
new thread is started. This thread starts the interactive PVS process and establishes a socket connection (like the batch TPC peer upon initialization). After that, it loops and receives messages over the socket connection: There is a Emacs function that sends proof status information of PVS to the interactive TPC peer. When the interactive TPC peer receives status information (which consists of the name of a lemma together with its proof status), it passes the information to the proof container.

**Theory Structure.** Each JIVE project is associated with a universal specification that contains

- the program-independent theories for the formal data and state model of SVENJA; these theories are part of the JIVE sources.

- the program-dependent theories that formalize properties of the input program and its specification; these theories are generated by PIS when a project is opened. They are located in the same directory as the input program.

- user-defined theories for the project that contain the universal specification (e.g., abstract data type definitions); these theories are part of the input to the JIVE system (like the program and the interface specification). They have to be located in the same directory as the input program.

The universal specification of a project with input program `prog.anja` consists of the following theories:

<table>
<thead>
<tr>
<th>Theory</th>
<th>File</th>
<th>Origin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>prog</code></td>
<td><code>prog.pvs</code></td>
<td>user-defined</td>
<td>the universal specification for the project. This theory contains the PVS declarations that are contained in comments of the form <code>/*PVS ... */</code> in a SVENJA program. The theory is automatically generated by the scanner.</td>
</tr>
<tr>
<td><code>TypeDefs</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>SubtypeEnum</code></td>
<td><code>progPrelude.pvs</code></td>
<td>generated</td>
<td>formalizes type ids</td>
</tr>
<tr>
<td><code>Attributes</code></td>
<td><code>progPrelude.pvs</code></td>
<td>generated</td>
<td>axiomatizes the subtype relation</td>
</tr>
<tr>
<td><code>progPrelude</code></td>
<td><code>progPrelude.pvs</code></td>
<td>generated</td>
<td>describes names and types of attributes</td>
</tr>
<tr>
<td><code>progTheory</code></td>
<td><code>progPrelude.pvs</code></td>
<td>generated</td>
<td>combines the predefined and the generated theories</td>
</tr>
<tr>
<td><code>JavaIntegers</code></td>
<td><code>JavaIntegers.pvs</code></td>
<td>predefined</td>
<td>contains the conjunction of the invariants</td>
</tr>
<tr>
<td><code>JavaTypes</code></td>
<td><code>JavaTypes.pvs</code></td>
<td>predefined</td>
<td>formalizes finite integers</td>
</tr>
<tr>
<td><code>Locations</code></td>
<td><code>Locations.pvs</code></td>
<td>predefined</td>
<td>formalizes types</td>
</tr>
<tr>
<td><code>StoreProperties</code></td>
<td><code>StoreProperties.pvs</code></td>
<td>predefined</td>
<td>formalizes locations</td>
</tr>
<tr>
<td><code>Stores</code></td>
<td><code>Stores.pvs</code></td>
<td>predefined</td>
<td>contains properties of object stores; in particular, lemmas from [PH97b]</td>
</tr>
</tbody>
</table>

24
### 5.1.2 Static View

The code for the TPC is located in directory jive/TPC. It consists of three major parts: The implementation of the peers, the LISP functions to control PVS, and the PVS theories. We describe these parts and the message format for the communication between the batch peer and the batch PVS in the following.

**The Peers.** The TPC peers are implemented in package jive.TPC.TPCTPCEer. It contains the following classes:
The Message Format. Messages consist of a command string and a set of argument strings. Currently, the following commands are supported:

<table>
<thead>
<tr>
<th>Command</th>
<th>Short Description</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>acknowledgment</td>
<td>—</td>
<td>is returned by Emacs after command was successfully executed.</td>
</tr>
<tr>
<td>ERR</td>
<td>error</td>
<td>error message</td>
<td>is returned by Emacs if execution of command leads to an error.</td>
</tr>
<tr>
<td>SD</td>
<td>shutdown</td>
<td>—</td>
<td>is sent to Emacs to shut down PVS and Emacs.</td>
</tr>
<tr>
<td>ALIVE</td>
<td>liveness test</td>
<td>—</td>
<td>is sent to Emacs to test whether the Emacs process is running and whether the socket connection works.</td>
</tr>
<tr>
<td>TCF</td>
<td>typecheck formula</td>
<td>formula, theory</td>
<td>is sent to Emacs to type check a formula in the context of a theory.</td>
</tr>
<tr>
<td>TCPI</td>
<td>typecheck prove import chain</td>
<td>theory</td>
<td>is sent to Emacs to type check a theory and its whole import chain.</td>
</tr>
<tr>
<td>CC</td>
<td>change context</td>
<td>context</td>
<td>is sent to Emacs to change the PVS context to the given directory.</td>
</tr>
</tbody>
</table>

To send messages over the socket connection, they are transformed into a string that contains (1) the command, (2) the number of arguments, and (3) the arguments. The different parts are enclosed in backslash characters\(^1\) and separated by blanks. For instance, a marshaled error message could look as follows: `\ERR\ 1 \This is an error message\.`

**LISP Functionality.** On the Emacs side, LISP code implements the functionality for controlling PVS and communicating with the peers. The LISP code resides in `jive/TPC/lisp`. For the batch TPC, the code is located in file `tpc.el`. The functions there perform the following tasks:

1. initialization of PVS and the socket connection,

\(^1\)Backslash characters must not appear in the arguments.
2. decoding messages,
3. invoking the required PVS functions, and
4. sending replies.

For the interactive TPC, the LISP code is located in file tpcInteractive.el. Since the communication is much simpler than with the batch TPC, only the following functionality is implemented there:

1. initialization of PVS and the socket connection, and
2. sending the proof status of a theory to the interactive peer (function jive-send-status).

**PVS Theories.** The program-independent theories are located in the directory jive/TPC/theories. Most parts of the theories are straightforward formalizations of the data and state model described in [PH97b]. However, two aspects require explanation.

**Formalization of Program Information.** To simplify reasoning (in particular, to benefit from automatically generated axioms), we formalize type and attribute ids as abstract data types. However, since abstract data types have a fixed number of constructors, this definition works for closed programs only (it is for instance possible to prove properties for all type ids by structural induction). Open programs require a different formalization of program information based on constants instead of data types.

**Theory Structure.** Logically, the program-independent theories for types, values, etc. build on the generated theories for type ids, attributes, etc. (in particular on the sorts defined by the data type declarations in the generated theories). However, since the names and locations of the generated theories vary, the predefined theories cannot import the generated theories. We use parameterized theories to avoid this problem: The predefined theories take the sorts, functions, etc. of the generated theories as theory parameters. This way, the generated theories (in particular, *progPrelude*) can import the predefined theories and instantiate them with the generated sorts and functions, yielding the theory structure shown in Fig. 5.2.
Chapter 6

Conclusions

In this report, we have described the implementation of the Java Interactive Verification Environment JIVE. To evaluate the quality of the tool, we have verified a nontrivial implementation of a doubly linked list w.r.t. an interface specification [LMMPH00]. The case study revealed several directions for enhancements of the JIVE system:

- The case study revealed that much of the effort is used for almost trivial, recurring proof steps, in particular for the verification of method invocations. This effort can be drastically reduced by elaborate proof strategies that develop large parts of the proofs automatically. So far, we implemented a weakest-precondition strategy, and strategies to verify virtual methods and to handle/eliminate assumptions [MPH00b]. The development of other strategies, for instance verification of method invocations is considered further work. Since prove strategies are loaded dynamically at system startup (see Chapter 4), new strategies can be integrated without modifying existing code.

- In the current implementation of JIVE, we exploit subtyping in PVS specifications. Since the PVS type system is undecidable, that leads to large numbers of type check conditions that cause much of the proof effort for the program-independent lemmas. Thus, it seems reasonable to replace the current formalization of the data and state model by a version without subtyping. We will do that in future versions of JIVE.

- We want to extend the Java subset supported by the system. In the next release, we aim at covering full Java Card [Sun97]. Such an extension will require major modifications in the PIS component. Due to the modular structure of the PVC (see Chapter 4), extensions of the underlying programming logic will not require refactorings here.

- We are currently working on a new version of the MAX tool that is able to generate Java code. That will allow us to get rid of the Java Native Interface, which will make the code much easier to read and debug.

- In the long run, we want to support Isabelle [Pau94] as theorem prover component. Although the TPC peer allows us to attach different theorem provers to the system, switching to Isabelle will make major modifications necessary. In particular, we will have to change the formula syntax (which affects the PIS and the ANTLR formula parser) and re-implement the theories of the universal specification (both the predefined theories and the generated program-dependent theories).
Bibliography


Appendix A

Package jive.PC.Program

Package Contents

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompRefParserTokenType</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>CompRefTokenType</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignStatement</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>BlockStatement</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>CallStatement</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>CastStatement</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>CompRef</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>EmptyStatement</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>FieldReadStatement</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>FieldWriteStatement</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>IfStatement</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>CompRefLexer</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Lexer</td>
</tr>
<tr>
<td>InvocationStatement</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>MethodRef</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>ReturnStatement</td>
<td>41</td>
</tr>
</tbody>
</table>
...no description...
Statement ................................................................. 41
...no description...
StatementList .......................................................... 43
...no description...
StaticInvocationStatement ............................................ 46
...no description...
UniversalInvocation ..................................................... 46
...no description...
VirtualMethod ............................................................ 47
...no description...
WhileStatement .......................................................... 48
...no description...
CompRefParser ............................................................ 49
Parser
Implementation ............................................................ 50
...no description...

A.1 Interfaces

A.1.1 Interface CompRefParserTokenType

A.1.1.1 Declaration

public interface CompRefParserTokenType

A.1.1.2 Fields

- public static final int EOF
- public static final int NULL_TREE_LOOKAHEAD
- public static final int AT
- public static final int COLON
- public static final int COMMA
• public static final int LPAR
  –
• public static final int RPAR
  –
• public static final int LSB
  –
• public static final int RSB
  –
• public static final int ID
  –
• public static final int NUMBER
  –
• public static final int WS
  –

A.1.2 Interface CompRefTokenType

A.1.2.1 Declaration

public interface CompRefTokenType

A.1.2.2 Fields

• public static final int EOF
  –
• public static final int NULL_TREE_LOOKAHEAD
  –
• public static final int AT
  –
• public static final int COLON
  –
• public static final int COMMA
  –
• public static final int LPAR
- public static final int RPAR
  -
- public static final int LSB
  -
- public static final int RSB
  -
- public static final int ID
  -
- public static final int NUMBER
  -
- public static final int WS
  -

A.2 Classes

A.2.1 Class AssignStatement

A.2.1.1 Declaration

```java
public class AssignStatement
    extends jive.PC.Program.Statement
```

A.2.1.2 Constructors

- `AssignStatement`
  public `AssignStatement`( jive.PC.Program.Implementation i, int o )

  - Usage
    * constructor with parameters for all attributes we assume, that the
      parameters point to a valid CompRef
A.2.1.3 Methods

- `getLHS`
  public String getLHS( )
  
  - Usage
  * return left-hand-side of assignment

- `getRHS`
  public String getRHS( )
  
  - Usage
  * return right-hand-side of assignment

A.2.2 Class BlockStatement

A.2.2.1 Declaration

| public class BlockStatement |
| extends jive.PC.Program.Statement |

A.2.2.2 Constructors

- `BlockStatement`
  public BlockStatement( jive.PC.Program.Implementation i, int o )
  
  - Usage
  * constructor with parameters for all attributes we assume, that the
    parameters point to a valid CompRef

A.2.2.3 Methods

- `getBody`
  public StatementList getBody( )
  
  - Usage
  * return body of block as StatementList

- `getEncIImplementation`
  public Implementation getEncIImplementation( )
  
  - Usage
  * returns the enclosing implementation of this

- `getLastStatement`
  public CompRef getLastStatement( )
- **Usage**
  * returns the textual last statement if a statement is composed of more than one statement

- **toString**
  public String toString()

### A.2.3 Class CallStatement

#### A.2.3.1 Declaration

```java
public class CallStatement
    extends jive.PC.Program.UniversalInvocation
```

#### A.2.3.2 Constructors

- **CallStatement**
  public CallStatement( jive.PC.Program.Implementation i, int o )
  
  - **Usage**
    * constructor with parameters for all attributes we assume, that the parameters point to a valid CompRef

#### A.2.3.3 Methods

- **getReverseSubstPairs**
  public HasTable getReverseSubstPairs()
  
  - **Usage**
    * prepare a table of substitutions [formPar/actPar] Here, we only add this

- **getSubstPairs**
  public HasTable getSubstPairs()
  
  - **Usage**
    * prepare a table of substitutions [actPar/formPar] Here, we only add this

- **getTarget**
  public String getTarget()
  
  - **Usage**
    * return target expression

### A.2.4 Class CastStatement
A.2.4.1 Declaration

```java
public class CastStatement
extends jive.PC.Program.AssignStatement
```

A.2.4.2 Constructors

- `CastStatement`
  ```java
  public CastStatement( jive.PC.Program.Implementation i, int o )
  ```
  - Usage
    * constructor with parameters for all attributes we assume, that the parameters point to a valid CompRef

A.2.4.3 Methods

- `getCastType`
  ```java
  public String getCastType( )
  ```
  - Usage
    * return right-hand-side of assignment as a type-String

A.2.5 Class CompRef

A.2.5.1 Declaration

```java
public abstract class CompRef
extends java.lang.Object
```

A.2.5.2 Constructors

- `CompRef`
  ```java
  public CompRef( )
  ```

A.2.5.3 Methods

- `checkCompRefEqual`
  ```java
  public void checkCompRefEqual( jive.PC.Program.CompRef crb,
  java.lang.String text )
  ```

- `checkInstanceOf`
  ```java
  public void checkInstanceOf( java.lang.Class c )
  ```
  - Usage
    * checks whether this is an instance of c
• `checkInstanceOf`
  public void checkInstanceOf( java.lang.Class c1, java.lang.Class c2 )

  - Usage
    * checks whether this is an instance of c1, or c2

• `getCRString`
  public CompRefString getCRString( )

• `getId`
  public Integer getId( )

• `getParentStatement`
  public Statement getParentStatement( )

  - Usage
    * helper method, returns the parent of a statement if possible

• `getStringRep`
  public String getStringRep( )

• `read`
  public static CompRef read( java.io.BufferedReader br )

  - Usage
    * reads a CompRef from the given BufferedReader. The formula is assumed to
      be correct

• `write`
  public void write( java.io.PrintWriter sw )

  - Usage
    * writes this CompRef into the PrintWriter

A.2.6 Class EmptyStatement

A.2.6.1 Declaration

public class EmptyStatement
  extends jive.PC.Program.Statement

A.2.6.2 Constructors

• `EmptyStatement`
  public EmptyStatement( jive.PC.Program.Implementation i, int o )

  - Usage
    * constructor with parameters for all attributes we assume, that the
      parameters point to a valid CompRef
A.2.6.3 Methods

- `toString`
  
  ```java
  public String toString()
  ```

A.2.7 `Class FieldReadStatement`

A.2.7.1 Declaration

```java
public class FieldReadStatement extends jive.PC.Program.Statement
```

A.2.7.2 Constructors

- `FieldReadStatement`
  
  ```java
  public FieldReadStatement( jive.PC.Program.Implementation i, int o )
  ```

  - Usage
    
    * constructor with parameters for all attributes we assume, that the parameters point to a valid CompRef

A.2.7.3 Methods

- `getAttribute`
  
  ```java
  public String getAttribute()
  ```

  - Usage
    
    * return attribute to be read as qualified name (T?att)

- `getLHS`
  
  ```java
  public String getLHS()
  ```

  - Usage
    
    * return left-hand-side

- `getTarget`
  
  ```java
  public String getTarget()
  ```

  - Usage
    
    * return target expression

A.2.8 `Class FieldWriteStatement`
A.2.8.1 Declaration

```java
public class FieldWriteStatement
    extends jive.PC.Program.Statement
```

A.2.8.2 Constructors

- `FieldWriteStatement`
  ```java
  public FieldWriteStatement(jive.PC.Program.Implementation i, int o)
  ```
  
  - Usage
    * constructor with parameters for all attributes we assume, that the
      parameters point to a valid CompRef

A.2.8.3 Methods

- `getAttribute`
  ```java
  public String getAttribute()
  ```
  
  - Usage
    * return attribute to be written as qualified name (T?att)

- `getRHS`
  ```java
  public String getRHS()
  ```
  
  - Usage
    * return right-hand-side

- `getTarget`
  ```java
  public String getTarget()
  ```
  
  - Usage
    * return target expression

A.2.9 Class IfStatement

A.2.9.1 Declaration

```java
public class IfStatement
    extends jive.PC.Program.Statement
```
A.2.9.2 Constructors

- **IfStatement**
  
  ```java
  public IfStatement( jive.PC.Program.Implementation i, int o )
  ```
  
  *Usage*
  * constructor with parameters for all attributes we assume, that the parameters point to a valid CompRef

A.2.9.3 Methods

- **getElseStatement**
  
  ```java
  public Statement getElseStatement( )
  ```
  
  *Usage*
  * return else-branch

- **getExpression**
  
  ```java
  public BooleanFormula getExpression( )
  ```

- **getLastStatement**
  
  ```java
  public CompRef getLastStatement( )
  ```

- **getPVSExpression**
  
  ```java
  public BooleanFormula getPVSExpression( )
  ```
  
  *Usage*
  * return guard of if in the form aB(EXP)

- **getThenStatement**
  
  ```java
  public Statement getThenStatement( )
  ```
  
  *Usage*
  * return then-branch

A.2.10 Class CompRefLexer

Lexer

A.2.10.1 Declaration

```java
public class CompRefLexer
  extends antlr.CharScanner
  implements CompRefTokenTypes, antlr.TokenStream
```
A.2.10.2 Constructors

- `CompRefLexer`
  public `CompRefLexer( antlr.InputBuffer ib )`

- `CompRefLexer`
  public `CompRefLexer( java.io.InputStream in )`

- `CompRefLexer`
  public `CompRefLexer( antlr.LexerSharedInputState state )`

- `CompRefLexer`
  public `CompRefLexer( java.io.Reader in )`

A.2.10.3 Methods

- `mAT`
  public `final void mAT( boolean _createToken )`

- `mCOLON`
  public `final void mCOLON( boolean _createToken )`

- `mCOMMA`
  public `final void mCOMMA( boolean _createToken )`

- `mID`
  public `final void mID( boolean _createToken )`

- `mLPAR`
  public `final void mLPAR( boolean _createToken )`

- `mLSB`
  public `final void mLSB( boolean _createToken )`

- `mNUMBER`
  public `final void mNUMBER( boolean _createToken )`

- `mRPAR`
  public `final void mRPAR( boolean _createToken )`

- `mRSB`
  public `final void mRSB( boolean _createToken )`

- `mWS`
  public `final void mWS( boolean _createToken )`

- `nextToken`
  public `Token nextToken( )`

A.2.11 Class InvocationStatement
A.2.11 Declaration

public class InvocationStatement
extends jive.PC.Program.UniversalInvocation

A.2.11.2 Constructors

- InvocationStatement
  public InvocationStatement( jive.PC.Program.Implementation i, int o )

  - Usage
    * constructor with parameters for all attributes we assume, that the
    parameters point to a valid CompRef

A.2.11.3 Methods

- getReverseSubstPairs
  public Hashtable getReverseSubstPairs( )

  - Usage
    * prepare a table of substitutions [formPar/actPar] Here, we only add this

- getSubstPairs
  public Hashtable getSubstPairs( )

  - Usage
    * prepare a table of substitutions [actPar/formPar] Here, we only add this

- getTarget
  public String getTarget( )

  - Usage
    * return target expression

A.2.12 Class MethodRef

A.2.12.1 Declaration

public abstract class MethodRef
extends jive.PC.Program.CompRef
A.2.12.2 Methods

- **equals**
  
  public boolean equals( java.lang.Object c )
  
  - Usage
    
    * structural equality

- **getMethodName**
  
  public String getMethodName( )
  
  - Usage
    
    * returns the m of T@m and T:m as String

- **getParameterCount**
  
  public int getParameterCount( )
  
  - Usage
    
    * return the num of parameters of this method

- **getParameterTypeAt**
  
  public String getParameterTypeAt( int i )
  
  - Usage
    
    * returns the type of the i-th parameter

- **getParameterTypes**
  
  public String getParameterTypes( )
  
  - Usage
    
    * returns the parameter types

- **getTypeExpression**
  
  public String getTypeExpression( )
  
  - Usage
    
    * returns at(T), it(T) or ct(T) of T@m and T:m as String we know that T is a legal type name

- **getTypeName**
  
  public String getTypeName( )
  
  - Usage
    
    * returns the T of T@m and T:m as String

- **hashCode**
  
  public int hashCode( )
  
  - Usage
jive.PC.Program– Statement

* hashcode for lookup table

• isAbstract
  public boolean isAbstract( )

  – Usage
  * check whether this points to an abstract method, to be overridden in VirtualMethod

• isFormPar
  public boolean isFormPar( java.lang.String s )

• toString
  public String toString( java.lang.String sep )

A.2.13 Class ReturnStatement

A.2.13.1 Declaration

public class ReturnStatement
extends jive.PC.Program.Statement

A.2.13.2 Constructors

• ReturnStatement
  public ReturnStatement( jive.PC.Program.Implementation i, int o )

  – Usage
  * constructor with parameters for all attributes we assume, that the parameters point to a valid CompRef

A.2.13.3 Methods

• getExpression
  public String getExpression( )

  – Usage
  * return expression to be returned

A.2.14 Class Statement

A.2.14.1 Declaration

public abstract class Statement
extends jive.PC.Program.CompRef
A.2.14.2 Fields

- public Implementation impl

- public int occurrence

A.2.14.3 Constructors

- Statement
  protected Statement( jive.PC.Program.Implementation i, int o )

  - Usage
    * constructor with parameters for all attributes we assume, that the
      parameters point to a valid CompRef

A.2.14.4 Methods

- equals
  public boolean equals( java.lang.Object c )

  - Usage
    * structural equality

- getImpl
  public Implementation getImpl( )

- getLast
  public Statement getLast( )

  - Usage
    * returns the textual last statement if a statement is composed of more than
      one statement

- getOccurrence
  public int getOccurrence( )

- getParent
  public Statement getParent( )

  - Usage
    * return parent statement e.g., the enclosing block, if, or while

- getParentStatement
  public Statement getParentStatement( )
- **Usage**
  * helper method, returns the parent of a statement if possible

- **getStringRep**
  ```
public String getStringRep() {
}
```

- **hashCode**
  ```
public int hashCode() {
}
```

- **Usage**
  * hashcode for lookup table

- **toString**
  ```
public String toString() {
}
```

### A.2.15 Class StatementList

#### A.2.15.1 Declaration
```java
public class StatementList
extends jive.PC.Program.CompRef
```

#### A.2.15.2 Fields

- public Implementation impl
  ```
  ...
  ```

- public int first
  ```
  ...
  ```

- public int last
  ```
  ...
  ```

#### A.2.15.3 Constructors

- **StatementList**
  ```java
  public StatementList( jive.PC.Program.Implementation i, int f, int l )
  ```
  - **Usage**
    * constructor with parameters for all attributes we assume, that the
      parameters point to a valid CompRef

- **StatementList**
  ```java
  public StatementList( jive.PC.Program.Statement s )
  ```
- **Usage**
  * constructs a list with one element s

---

**StatementList**

```java
public StatementList( jive.PC.Program.Statement a, jive.PC.Program.Statement b )
```

### A.2.15.4 Methods

- **conc**
  ```java
  public StatementList conc( jive.PC.Program.StatementList s )
  ```

  - **Usage**
    * concatenates this and s

- **contains**
  ```java
  public boolean contains( jive.PC.Program.Statement s )
  ```

  - **Usage**
    * yields true if this statementlist contains cr

- **equals**
  ```java
  public boolean equals( java.lang.Object c )
  ```

  - **Usage**
    * structural equality

- **getFirst**
  ```java
  public Statement getFirst( )
  ```

  - **Usage**
    * returns first statement

- **getHead**
  ```java
  public StatementList getHead( )
  ```

  - **Usage**
    * return list without last statement

- **getImpl**
  ```java
  public Implementation getImpl( )
  ```

  - **Usage**
    * returns the implementation which contains this statementlist

- **getLast**
  ```java
  public Statement getLast( )
  ```

  - **Usage**
* returns last statement

```java
public int getLength()
```

* getParent

```java
public Statement getParent()
```

- Usage
  * return enclosing block statement

* getParentStatement

```java
public Statement getParentStatement()
```

- Usage
  * helper method, returns the parent of a statement if possible

* getSingleStatement

```java
public ComRef getSingleStatement()
```

- Usage
  * returns the single compref if the statememlif contains only one element

* getStatements

```java
public Statement getStatements()
```

* getStringRep

```java
public String getStringRep()
```

- Usage
  * return a string representation of this statementlist

* getTail

```java
public StatementList getTail()
```

- Usage
  * return list without first statement

* hashCode

```java
public int hashCode()
```

- Usage
  * hashcode for lookup table

* isSingleStatement

```java
public boolean isSingleStatement()
```

- Usage
  * checks whether list has only one element
• isSuccessor
  public boolean isSuccessor( jive.PC.Program.StatementList s )

  – Usage
  * tests whether s follows this in the program

• toString
  public String toString( )

  – Usage
  * return unparsing of statement list

A.2.16 Class StaticInvocationStatement

A.2.16.1 Declaration

public class StaticInvocationStatement
extends jive.PC.Program.UniversalInvocation

A.2.16.2 Constructors

• StaticInvocationStatement
  public StaticInvocationStatement( jive.PC.Program.Implementation i, int o )

  – Usage
  * constructor with parameters for all attributes we assume, that the
  parameters point to a valid CompRef

A.2.17 Class UniversalInvocation

A.2.17.1 Declaration

public abstract class UniversalInvocation
extends jive.PC.Program.Statement

A.2.17.2 Constructors

• UniversalInvocation
  public UniversalInvocation( jive.PC.Program.Implementation i, int o )

  – Usage
  * constructor with parameters for all attributes we assume, that the
  parameters point to a valid CompRef
A.2.17.3 Methods

- getLHS
  public ProgVarString getLHS( )
    - Usage
      * return left-hand-side of invocation

- getMethod
  public MethodRef getMethod( )
    - Usage
      * return method to be invoked

- getReverseSubstPairs
  public Hashtable getReverseSubstPairs( )
    - Usage
      * prepare a table of substitutions [form/actPar] Here, we do not handle the implicit parameter since this might be a static invocation.

- getSubstPairs
  public Hashtable getSubstPairs( )
    - Usage
      * prepare a table of substitutions [actPar/formPar] Here, we do not handle the implicit parameter since this might be a static invocation.

A.2.18 Class VirtualMethod

A.2.18.1 Declaration

    public class VirtualMethod
    extends jive.PC.Program.MethodRef

A.2.18.2 Constructors

- VirtualMethod
    - Usage
      * constructor with parameters for all attributes we assume, that the parameters point to a valid CompRef
A.2.18.3 Methods

- **checkIsNotAbstract**
  public void checkIsNotAbstract() 
  - Usage
    * checks whether this method is abstract and throws aReqException is not

- **getImplementation**
  public Implementation getImplementation() 
  - Usage
    * returns the implementation associated with the virtual method. The result is 
      not defined for abstract methods.

- **isAbstract**
  public boolean isAbstract() 
  - Usage
    * check whether this points to an abstract method

- **overrides**
  public boolean overrides( jive.PC.Program.VirtualMethod vm ) 
  - Usage
    * checks whether this overrides vm

- **toString**
  public String toString() 
  - Usage
    * returns a textual representation of the implementation T@m(TP1, ..., TPn)

A.2.19 Class WhileStatement

A.2.19.1 Declaration

```java
public class WhileStatement
    extends jive.PC.Program.Statement
```

A.2.19.2 Constructors

- **WhileStatement**
  public WhileStatement( jive.PC.Program.Implementation i, int o ) 
  - Usage
    * constructor with parameters for all attributes we assume, that the 
      parameters point to a valid CompRef
A.2.19.3 Methods

- `getBody`
  public `Statement getBody()`
  
  Usage
  * return loop body

- `getExpression`
  public `BooleanFormula getExpression()`

- `getLastStatement`
  public `CompRef getLastStatement()`

- `getPVSExpression`
  public `BooleanFormula getPVSExpression()`
  
  Usage
  * return guard of if in the form `aB(EXP)`

A.2.20 Class `CompRefParser`

Parser

A.2.20.1 Declaration

```java
public class CompRefParser
extends antlr.LLkParser
implements CompRefParserTokenTypes
```

A.2.20.2 Fields

- public `String method`
  
- public `String classname`
  
- public `boolean impl`
  
- public `Vector par`
  
- public `int stmt1`
public int stmt2
-

public static final String tokenNames
-

A.2.20.3 Constructors

- CompRefParser
  public CompRefParser( antlr.ParserSharedInputState state )

- CompRefParser
  public CompRefParser( antlr.TokenBuffer tokenBuf )

- CompRefParser
  protected CompRefParser( antlr.TokenBuffer tokenBuf, int k )

- CompRefParser
  public CompRefParser( antlr.TokenStream lexer )

- CompRefParser
  protected CompRefParser( antlr.TokenStream lexer, int k )

A.2.20.4 Methods

- compref
  public final void compref( )

- goal
  public final void goal( )

- parlist
  public final void parlist( )

- statements
  public final void statements( )

A.2.21 Class Implementation

A.2.21.1 Declaration

```java
public class Implementation
extends jive.PC.Program.MethodRef
```
A.2.21.2 Constructors

- *Implementation*
  public Implementation( java.lang.String cl, java.lang.String me, java.lang.String[] pt )

A.2.21.3 Methods

- *getBody*
  public BlockStatement getBody( )
    - Usage
      * return body of implementation as block

- *getInitialization*
  public BooleanFormula getInitialization( )
    - Usage
      * return a Formula describing the initialization of local variables /“vi = init(TVi)"

- *getLocalVariables*
  public String getLocalVariables( )
    - Usage
      * returns the local variables of this implementation

- *toString*
  public String toString( )
    - Usage
      * returns a textual representation of the implementation T@m(TP1, ..., TPn)
Appendix B

Package jive

Package Contents

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtensionFileFilter</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>Jive</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
</tbody>
</table>

B.1 Classes

B.1.1 Class ExtensionFileFilter

B.1.1.1 Declaration

```java
public class ExtensionFileFilter
extends javax.swing.filechooser.FileFilter
```

B.1.1.2 Constructors

- `ExtensionFileFilter`
  ```java
  public ExtensionFileFilter( java.lang.String ex, java.lang.String d )
  ```

B.1.1.3 Methods

- `accept`
  ```java
  public boolean accept( java.io.File f )
  ```
- `getDescription`
  ```java
  public String getDescription( )
  ```
B.1.2 **Class Jive**

### Declaration

```java
public class Jive
extends java.lang.Object
```

### Fields

- public static String jiveHome
  
- public static boolean runPVS
  
### Constructors

- **Jive**
  ```java
  public Jive()
  ```

### Methods

- `chooseFile`
  ```java
  public static File chooseFile( java.awt.Component parent, java.swing.filechooser.FileFilter filter )
  ```

- `logMsg`
  ```java
  public static void logMsg( java.lang.String s )
  ```
  - **Usage**
    * prints message to log file
  - **Parameters**
    * s - string to be printed

- `main`
  ```java
  public static void main( java.lang.String [] argv )
  ```

- `newProject`
  ```java
  public static void newProject()
  ```

- `openControlFrame`
  ```java
  protected static void openControlFrame()
  ```

- `printErr`
  ```java
  public static void printErr( java.lang.String s )
  ```
- **Usage**
  * prints error message

- **Parameters**
  * s - string to be printed

---

- **printMsg**
  
  public static void printMsg( java.lang.String s )

  - **Usage**
    * prints message

  - **Parameters**
    * s - string to be printed

---

- **shutdown**
  
  public static void shutdown( )
Appendix C

Package jive.PVC.Container.View

<table>
<thead>
<tr>
<th>Package Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td></td>
</tr>
<tr>
<td>AxiomMenuItem</td>
<td>56</td>
</tr>
<tr>
<td>BackwardMenuItem</td>
<td>56</td>
</tr>
<tr>
<td>FormulaLine</td>
<td>56</td>
</tr>
<tr>
<td>ForwardMenuItem</td>
<td>57</td>
</tr>
<tr>
<td>MenuItemLoadException</td>
<td>57</td>
</tr>
<tr>
<td>View</td>
<td>57</td>
</tr>
<tr>
<td>TreeViewEditor</td>
<td>59</td>
</tr>
<tr>
<td>OperationMenuItem</td>
<td>60</td>
</tr>
<tr>
<td>PostCondition</td>
<td>60</td>
</tr>
<tr>
<td>TacticLoadException</td>
<td>61</td>
</tr>
<tr>
<td>TacticMenuItem</td>
<td>61</td>
</tr>
<tr>
<td>PreCondition</td>
<td>61</td>
</tr>
<tr>
<td>TreeViewKeyListener</td>
<td>62</td>
</tr>
<tr>
<td>TreeViewRenderer</td>
<td>62</td>
</tr>
<tr>
<td>EmptyLine</td>
<td>63</td>
</tr>
<tr>
<td>ProgressWindow</td>
<td>63</td>
</tr>
</tbody>
</table>
C.1 Classes

C.1.1 Class AxiomMenuItem

C.1.1.1 Declaration

```java
public class AxiomMenuItem
extends jive.PVC.Container.View.OperationMenuItem
```

C.1.1.2 Constructors

- `AxiomMenuItem`
  ```java
  public AxiomMenuItem( jive.PVC.Container.ProofContainer c,
                          java.lang.Class cl, jive.PVC.Container.View.View tvp )
  ```

C.1.2 Class BackwardMenuItem

C.1.2.1 Declaration

```java
public final class BackwardMenuItem
extends jive.PVC.Container.View.OperationMenuItem
```

C.1.2.2 Constructors

- `BackwardMenuItem`
  ```java
  public BackwardMenuItem( jive.PVC.Container.ProofContainer c,
                           java.lang.Class cl, jive.PVC.Container.View.View tvp )
  ```

C.1.3 Class FormulaLine

C.1.3.1 Declaration

```java
public abstract class FormulaLine
extends jive.PVC.Container.View.UnparseLine
```
C.1.3.2 Methods

- `toString`
  ```java
  public String toString()
  ```

C.1.4 Class ForwardMenuItem

C.1.4.1 Declaration
```
public final class ForwardMenuItem 
extends jive.PVC.Container.View.OperationMenuItem
```

C.1.4.2 Constructors

- `ForwardMenuItem`
  ```java
  public ForwardMenuItem( jive.PVC.Container.ProofContainer c, 
  java.lang.Class cl, jive.PVC.Container.View.View tvp )
  ```

C.1.5 Class MenuItemLoadException

C.1.5.1 Declaration
```
public class MenuItemLoadException 
extends java.lang.Exception
```

C.1.5.2 Constructors

- `MenuItemLoadException`
  ```java
  public MenuItemLoadException( java.lang.String s )
  ```

C.1.6 Class View

C.1.6.1 Declaration
```
public abstract class View 
extends javax.swing.JFrame
```
C.1.6.2 Fields

- public static final int LOAD
  - 
- public static final int SAVE
  - 

C.1.6.3 Methods

- activateNode
  public abstract void activateNode( javax.swing.tree.TreeNode t )
    - Usage
      * shows the node and activates it (whatever this means for a view)

- close
  protected void close( )

- doLoadSave
  public void doLoadSave( int mode )

- enableClose
  public abstract void enableClose( boolean b )
    - Usage
      * tells this container whether choosing exit in the file menu should be possible or not

- exit
  protected void exit( )

- getAllPtnSelections
  public ProofTreeNode getAllPtnSelections( )
    - Usage
      * return all selected ProofTreeNodes as an array

- getContainer
  public ProofContainer getContainer( )

- getFrame
  public abstract Frame getFrame( )
    - Usage
      * return the frame, needed for Dialogs
- **getParameter**
  ```java
  public Object getParameter( java.lang.String name,
      jive.PVC.Container.Rule rule )
  ```

- **getPtnSelection**
  ```java
  public ProofTreeNode getPtnSelection( )
  ```

- **getSelectedCompRef**
  ```java
  public CompRef getSelectedCompRef( )
  ```

- **message**
  ```java
  public abstract void message( java.lang.String text )
  ```
  - **Usage**
    ```java
    * all text messages to a view can be send by this method
    ```

- **nodeAdded**
  ```java
  public abstract void nodeAdded( javax.swing.tree.TreeNode t )
  ```
  - **Usage**
    ```java
    * the node t was added
    ```

- **nodeChanged**
  ```java
  public abstract void nodeChanged( javax.swing.tree.TreeNode t )
  ```
  - **Usage**
    ```java
    * the node t was added
    ```

- **nodeRemoved**
  ```java
  public abstract void nodeRemoved( javax.swing.tree.TreeNode parent,
      javax.swing.tree.TreeNode t )
  ```
  - **Usage**
    ```java
    * the node t was removed from parent parent
    ```

- **nodeStructureChanged**
  ```java
  public abstract void nodeStructureChanged( javax.swing.tree.TreeNode parent )
  ```
  - **Usage**
    ```java
    * the node t was removed from parent parent
    ```

### C.1.7 Class TreeViewEditor

#### C.1.7.1 Declaration

```java
public class TreeViewEditor
extends javax.swing.JPanel
implements javax.swing.tree.TreeCellEditor, java.awt.event.ActionListener
```
C.1.7.2 Methods

- `actionPerformed`
  ```java
  public void actionPerformed( java.awt.event.ActionEvent e )
  ```

- `addCellEditorListener`
  ```java
  public void addCellEditorListener( javax.swing.event.CellEditorListener l )
  ```

- `cancelCellEditing`
  ```java
  public void cancelCellEditing( )
  ```

- `getCellEditorValue`
  ```java
  public Object getCellEditorValue( )
  ```

- `getTreeCellEditorComponent`
  ```java
  public Component getTreeCellEditorComponent( javax.swing.JTree tree, java.lang.Object value, boolean selected, boolean expanded, boolean leaf, int row )
  ```

- `isCellEditable`
  ```java
  public boolean isCellEditable( java.util.EventObject anEvent )
  ```

- `removeCellEditorListener`
  ```java
  public void removeCellEditorListener( javax.swing.event.CellEditorListener l )
  ```

- `shouldSelectCell`
  ```java
  public boolean shouldSelectCell( java.util.EventObject anEvent )
  ```

- `stopCellEditing`
  ```java
  public boolean stopCellEditing( )
  ```

C.1.8 Class `OperationMenuItem`

C.1.8.1 Declaration

```java
public abstract class OperationMenuItem
  extends javax.swing.JMenu
  implements java.awt.event.ActionListener
```

C.1.8.2 Methods

- `actionPerformed`
  ```java
  public void actionPerformed( java.awt.event.ActionEvent ae )
  ```

C.1.9 Class `PostCondition`
C.1.9.1 Declaration

```java
public final class PostCondition
extends jive.PVC.Container.View.FormulaLine
```

C.1.10 Class TacticLoadException

C.1.10.1 Declaration

```java
public class TacticLoadException
extends java.lang.Exception
```

C.1.10.2 Constructors

- `TacticLoadException`
  ```java
  public TacticLoadException( java.lang.String s )
  ```

C.1.11 Class TacticMenuItem

C.1.11.1 Declaration

```java
public class TacticMenuItem
extends jive.PVC.Container.View.OperationMenuItem
```

C.1.11.2 Constructors

- `TacticMenuItem`
  ```java
  public TacticMenuItem( jive.PVC.Container.ProofContainer c,
  java.lang.Class cl, jive.PVC.Container.View.View tvp )
  ```

C.1.11.3 Methods

- `cutName`
  ```java
  public String cutName( java.lang.String s )
  ```

C.1.12 Class PreCondition

C.1.12.1 Declaration

```java
public final class PreCondition
extends jive.PVC.Container.View.FormulaLine
```
C.1.13  Class TreeViewKeyListener

C.1.13.1 Declaration

public class TreeViewKeyListener
extends java.lang.Object
implements java.awt.event.KeyListener

C.1.13.2 Constructors

- TreeViewKeyListener
  public TreeViewKeyListener( jive.PVC.Container.View.TreeView treeView )

C.1.13.3 Methods

- keyPressed
  public void keyPressed( java.awt.event.KeyEvent evt )
- keyReleased
  public void keyReleased( java.awt.event.KeyEvent evt )
- keyTyped
  public void keyTyped( java.awt.event.KeyEvent evt )

C.1.14  Class TreeViewRenderer

C.1.14.1 Declaration

public class TreeViewRenderer
extends java.lang.Object
implements javax.swing.tree.TreeCellRenderer

C.1.14.2 Constructors

- TreeViewRenderer
  public TreeViewRenderer( )
C.1.14.3 Methods

- `getTreeCellRendererComponent`
  ```java
  public Component getTreeCellRendererComponent( javax.swing.JTree tree, java.lang.Object value, boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus )
  ```

- `renderProofTreeNode`
  ```java
  protected Component renderProofTreeNode( javax.swing.JTree tree, jive.PVC.Container.ProofTreeNode ptn, boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus )
  ```

C.1.15 Class EmptyLine

C.1.15.1 Declaration

```java
public class EmptyLine
extends jive.PVC.Container.View.UnparseLine
```

C.1.15.2 Constructors

- `EmptyLine`
  ```java
  public EmptyLine( )
  ```

C.1.16 Class ProgressWindow

C.1.16.1 Declaration

```java
public class ProgressWindow
extends javax.swing.JFrame
implements java.awt.event.ActionListener
```

C.1.16.2 Constructors

- `ProgressWindow`
  ```java
  public ProgressWindow( java.lang.String s, java.lang.Object op, int minp, int maxp )
  ```

C.1.16.3 Methods

- `actionPerformed`
  ```java
  public void actionPerformed( java.awt.event.ActionEvent e )
  ```

- `main`
  ```java
  public static void main( java.lang.String [] argv )
  ```
C.1.17  Class TextView

C.1.17.1  Declaration

```java
public class TextView
extends jive.PVC.Container.View.View
```

C.1.17.2  Serializable Fields

- private String indent
- private boolean isImplementationUnparsed
- private int dividerLocation

C.1.17.3  Methods

- `activateNode`
  ```java
  public void activateNode( javax.swing.tree.TreeNode t )
  ```
  - Usage
    * shows the node and activates it (whatever this means for a view)

- `build_textarea`
  ```java
  protected void build_textarea( javax.swing.JSplitPane splitpane )
  ```

- `build_toolbar`
  ```java
  protected void build_toolbar( )
  ```

- `contains`
  ```java
  public boolean contains( jive.PC.Program.ComprRef cr )
  ```

- `createTextview`
  ```java
  public static void createTextview( jive.PVC.Container.ProofContainer pc, jive.PVC.Container.ProofTreeNode ptn )
  ```

- `enableClose`
  ```java
  public void enableClose( boolean b )
  ```
  - Usage
    * tells this container whether choosing exit in the file menu should be possible
    or not
• **getContainer**

  public ProofContainer getContainer()

  
  - **Usage**
  * return the container of this view

• **getFrame**

  public Frame getFrame()

• **message**

  public void message( java.lang.String text )

  
  - **Usage**
  * all text messages to a view can be send by this method

• **nodeAdded**

  public void nodeAdded( javax.swing.tree.TreeNode t )

  
  - **Usage**
  * the node t was added

• **nodeChanged**

  public void nodeChanged( javax.swing.tree.TreeNode t )

  
  - **Usage**
  * the node t was added

• **nodeRemoved**

  public void nodeRemoved( javax.swing.tree.TreeNode parent,
  javax.swing.tree.TreeNode t )

  
  - **Usage**
  * the node t was removed from parent parent

• **nodeStructureChanged**

  public void nodeStructureChanged( javax.swing.tree.TreeNode parent )

  
  - **Usage**
  * textview has not to care about this

• **setFrameProperties**

  protected void setFrameProperties()

C.1.18  **Class TreeView**

C.1.18.1  **Declaration**

```java
public class TreeView extends jive.PVC.Container.View.View
```
C.1.18.2 Constructors

- **TreeView**
  
  ```java
  public TreeView( jive.PVC.Container.ProofContainer c )
  ```

C.1.18.3 Methods

- **activateNode**
  ```java
  public void activateNode( javax.swing.tree.TreeNode t )
  ```

- **build_menus**
  ```java
  protected void build_menus( )
  ```

- **build_prooftree**
  ```java
  protected void build_prooftree( )
  ```

- **build_textarea**
  ```java
  protected void build_textarea( javax.swing.JSplitPane splitpane )
  ```

- **build_toolbar**
  ```java
  protected void build_toolbar( )
  ```

- **build_tree**
  ```java
  protected void build_tree( javax.swing.JSplitPane splitpane )
  ```

- **collapseNodes**
  ```java
  public void collapseNodes( boolean isGoal )
  ```

- **enableClose**
  ```java
  public void enableClose( boolean b )
  ```

  - **Usage**
    * tells this container whether it should be possible to select exit or not

- **getFrame**
  ```java
  public Frame getFrame( )
  ```

- **message**
  ```java
  public void message( java.lang.String text )
  ```

- **nodeAdded**
  ```java
  public void nodeAdded( javax.swing.tree.TreeNode t )
  ```

- **nodeChanged**
  ```java
  public void nodeChanged( javax.swing.tree.TreeNode t )
  ```

- **nodeRemoved**
  ```java
  public void nodeRemoved( javax.swing.tree.TreeNode parent,
  javax.swing.tree.TreeNode t )
  ```
• `nodeStructureChanged`
  
  ```java
  public void nodeStructureChanged( javax.swing.tree.TreeNode parent )
  ```

• `set_frame_properties`

  ```java
  protected void set_frame_properties( )
  ```
Appendix D

Package jive.PC.PCPeer

Package Contents

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IllegalCompRefException</td>
<td>68</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>68</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>SyntaxException</td>
<td>69</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>UndeclaredVariableException</td>
<td>69</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>PCPeer</td>
<td>70</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
</tbody>
</table>

D.1 Classes

D.1.1 Class IllegalCompRefException

D.1.1.1 Declaration

```java
public class IllegalCompRefException
extends java.lang.Exception
```

D.1.1.2 Constructors

- `IllegalCompRefException`
  ```java
  public IllegalCompRefException(java.lang.String s)
  ```

D.1.2 Class Node
D.1.2.1 Declaration

```java
public class Node
extends java.lang.Object
```

D.1.2.2 Fields

- public long sortIndex

- public long elementIndex

D.1.2.3 Constructors

- `Node`
  ```java
  public Node( long sort, long element )
  ```

D.1.3 Class `SyntaxException`

D.1.3.1 Declaration

```java
public class SyntaxException
extends java.lang.RuntimeException
```

D.1.3.2 Constructors

- `SyntaxException`
  ```java
  public SyntaxException( java.lang.String p )
  ```

D.1.4 Class `UndeclaredVariableException`

D.1.4.1 Declaration

```java
public class UndeclaredVariableException
extends jive.PC.PCPeer.SyntaxException
```

D.1.4.2 Constructors

- `UndeclaredVariableException`
  ```java
  public UndeclaredVariableException( java.lang.String p )
  ```
D.1.5  Class PCPeer

D.1.5.1 Declaration

```
public class PCPeer
extends java.lang.Object
```

D.1.5.2 Fields

- public static char CONCRETE
  -
- public static char ABSTRACT
  -
- public static char INTERFACE
  -
- public static HasTable comprefs
  -
- public static HasTable progVars
  -
- public static boolean MAX_RUNNING
  -

D.1.5.3 Constructors

- `PCPeer`
  public PCPeer( )

D.1.5.4 Methods

- `addComprefToHasTable`
  protected static void addComprefToHasTable( jive.PC.Program.MethodRef ref, java.lang.Object node, boolean sup )

- `analyzeProgram`
  protected static native boolean analyzeProgram( java.lang.String file, java.lang.String prelude )

  - Usage
    * JNI method invokes frontend on file
• checkVarRuleProgVar
  public static void checkVarRuleProgVar(
    jive.PVC.Container.ProgVarString name, jive.PC.Program.CompRef cr )

    – Usage
      * checks, if name is a progvar in the cr context if not it throws a ReqException

• checkVarRuleProgVar
  public static void checkVarRuleProgVar( java.lang.String name,
    jive.PC.Program.CompRef cr )

• createCompRef
  public static CompRef createCompRef( java.lang.String s )

    – Usage
      * returns the compref described by the given String s. If no such compref
        exists, an IllegalCompRefException is throw

• fillSymbolTable
  public static native void fillSymbolTable(  
    jive.PVC.Container.ProofContainer c )

• generateTheories
  protected static void generateTheories( java.lang.String path,
    java.lang.String basename )

    – Usage
      * generates program-dependent PVS theories

• getActualParameter
  public static native String getActualParameter( 
    jive.PC.Program.UniversalInvocation s, int i )

• getAttribute
  public static native String getAttribute( jive.PC.Program.Statement s )

• getAxiom
  protected static native String getAxiom( int i )

    – Usage
      * JNI method returns the i-th axiom generated by the frontend check
        pis_help.c for the mapping from i to axioms

• getBody
  public static int getBody( jive.PC.Program.Implementation i )

• getCastType
  public static native String getCastType( jive.PC.Program.Statement s )
• `getChildStmt`
  public static native int `getChildStmt(jive.PC.Program.Statement s, int i)`

• `getDeclType`
  public static String `getDeclType(jive.PC.Program.VirtualMethod ref)`

• `getDeclTypeHelp`
  protected static native String `getDeclTypeHelp(long sort, long index)`

• `getDirectSubtypes`
  public static native String `getDirectSubtypes(java.lang.String T)`
  - Usage
    * returns all directs subtypes of type T

• `getExpr`
  public static native String `getExpr(jive.PC.Program.Statement s)`

• `getFirstStmt`
  public static native int `getFirstStmt(jive.PC.Program.BlockStatement b)`

• `getFormalParameter`
  public static native String `getFormalParameter(jive.PC.Program.MethodRef ref, int i)`

• `getFormalParameterCount`
  public static native int `getFormalParameterCount(jive.PC.Program.MethodRef ref)`
  - Usage
    * returns the number of formal parameters of method ref

• `getFormalParameterNames`
  public static String `getFormalParameterNames(jive.PC.Program.MethodRef ref)`
  - Usage
    * returns the names of the formal Parameters as an array

• `getInvariants`
  protected static native String `getInvariants()`

• `getJavaType`
  public static String `getJavaType(java.lang.String type)`
  - Usage
    * returns the JavaType equivalent for type, i.e., booleanType, intType, 
      ct(type), at(type), or (it(type)
• getKindOfType
  public static native char getKindOfType( java.lang.String c )

• getLastStmt
  public static native int getLastStmt( jive.PC.Program.BlockStatement b )

• getLHS
  public static native String getLHS( jive.PC.Program.Statement s )

• getLocalVariables
  public static native String getLocalVariables( jive.PC.Program.Implementation i )

• getMethod
  public static native MethodRef getMethod( jive.PC.Program.UniversalInvocation s )

• getMethodRefs
  public static MethodRef getMethodRefs( )

  – Usage
    * returns all method implementation objects

• getParentStmt
  public static native int getParentStmt( jive.PC.Program.Statement s )

• getParentStmt
  public static native int getParentStmt( jive.PC.Program.StatementList s )

• getPredecessor
  public static native int getPredecessor( jive.PC.Program.Implementation i, int s )

• getProgVars
  public static native String getProgVars( jive.PC.Program.Implementation cr )

  – Usage
    * returns all local variables + parameters of implementation cr

• getReqClause
  protected static native String getReqClause( jive.PC.Program.MethodRef m )

• getSort
  public static String getSort( java.lang.String s, java.util.Hashtable st, jive.PC.Program.ComRef ref )

  – Usage
* return the sort of a variable if s is program variable and declared in ref:
  return Value or Store else look up sort in symbol table

- **getStatement**
  ```java
  public static native String getStatement(
      jive.PC.Program.Implementation ref, int oc )
  ```

- **getSuccessor**
  ```java
  public static native int getSuccessor( jive.PC.Program.Implementation i, int s )
  ```

- **getTarget**
  ```java
  public static native String getTarget( jive.PC.Program.Statement s )
  ```

- **getExpression**
  ```java
  public static String getExpression( java.lang.String type )
  
  - Usage
    * returns at(T), it(T), or ct(T) of T as String
  ```

- **getUnparsing**
  ```java
  public static native String getUnparsing( 
      jive.PC.Program.Implementation i, int s )
  ```

- **getVarRuleProgVars**
  ```java
  public String getVarRuleProgVars( jive.PC.Program.UniversalInvocation ui )
  ```

- **getVirtualMethods**
  ```java
  public static VirtualMethod getVirtualMethods( )
  
  - Usage
    * returns all virtual method objects of the underlying program
  ```

- **initCompressHashtable**
  ```java
  protected static void initCompressHashtable( )
  ```

- **insertCompress**
  ```java
  protected static native void insertCompress( java.util.Hashtable c )
  ```

- **insertSequences**
  ```java
  protected static void insertSequences(
      jive.PC.Container.ProofContainer c )
  
  - Usage
    * constructs all sequents about invariants. adds them as goal or not, according
to whether the method is native or not. This method must be called AFTER
the compref table was built!
• insertSequent
    
      Usage
      * helper for insertSequentsNative, insertInvSequents this method is called from C to introduce one Sequent (Axiom or Goal) into the ProofContainer

• insertSequents
  public static void insertSequents( jive.PC.Container.ProofContainer c )
    
      Usage
      * insert all sequents (stemming from pre-post pairs or invariants) as goal or normal sequent.

• insertSequentsNative
  protected static native void insertSequentsNative( jive.PC.Container.ProofContainer c )

• isAbstract
  public static native boolean isAbstract( jive.PC.Program.VirtualMethod ref )

• isDirectSubtype
  public static native boolean isDirectSubtype( java.lang.String S, java.lang.String T )
    
      Usage
      * returns true, iff S is a direct Subtype of T

• isLegalCompRef
  public static native boolean isLegalCompRef( jive.PC.Program.CompRef c )

• isLogicalVariable
  public static native boolean isLogicalVariable( java.lang.String s, jive.PC.Program.CompRef ref, java.util.Hashtable c )

• isNative
  protected static native boolean isNative( jive.PC.Program.MethodRef ref )

• isPrivate
  protected static native boolean isPrivate( jive.PC.Program.MethodRef ref )

• isProgVar
  public static boolean isProgVar( java.lang.String s, jive.PC.Program.CompRef ref )
- **Usage**
  * return whether s is a program variable in context ref

```
• isStaticMethod
public static native boolean isStaticMethod( jive.PC.Program.MethodRef ref )
```

- **Usage**
  * check whether ref is a static method

```
• isSubtype
public static native boolean isSubtype( java.lang.String S, java.lang.String T )
```

- **Usage**
  * checks whether S is a subtype of T

```
• isTypeName
public static native boolean isTypeName( java.lang.String s )
```

```
• isUsedAsProgVar
public static boolean isUsedAsProgVar( java.lang.String v )
```

- **Usage**
  * returns whether v is used as program variable anywhere in the program

```
• newProject
```

```
• println
public static void println( java.lang.Object o )
```

```
• st
public static void st( )
```
Appendix E

Package jive.TPC.TPCPeer

Package Contents

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCException</td>
<td>77</td>
</tr>
<tr>
<td>TPCPeer</td>
<td>77</td>
</tr>
<tr>
<td>InteractiveTPCPeer</td>
<td>79</td>
</tr>
</tbody>
</table>

This class hosts a server socket and waits for the TCP to connect.

E.1 Classes

E.1.1 Class TPCException

E.1.1.1 Declaration

```java
public class TPCException
extends java.lang.Exception
```

E.1.1.2 Constructors

- **TPCException**
  ```java
  public TPCException( java.lang.String m )
  ```

E.1.2 Class TPCPeer

E.1.2.1 Declaration

```java
public class TPCPeer
extends java.lang.Object
```
E.1.2.2 Constructors

- **TPCPeer**
  
  public TPCPeer( )

E.1.2.3 Methods

- **getReply**
  
  protected static Message getReply( int time )

  - **Usage**
    
    * wait for reply from PVS set timeout to time millisecs

- **init**
  
  public static void init( )

  - **Usage**
    
    * start TPC and initialize communication

- **main**
  
  public static void main( java.lang.String [] argv )

- **openProject**
  
  public static void openProject( java.lang.String path,
  java.lang.String basename )

  - **Usage**
    
    * prepare PVS to handle new project all program-dependent theories must
      exist upon invocation of this method change context to path
      typecheck-importchain whole prelude path: directory of new context
      basename: filename without extensions

- **sendMessage**
  
  protected static Message sendMessage( jive.TPC.TPCPeer.Message m,
  boolean timeout )

  - **Usage**
    
    * send Message to PVS, return reply

  - **Parameters**
    
    * m - Message to be sent
    * timeout - timeout for reply

- **shutdown**
  
  public static void shutdown( )

  - **Usage**
    
    * initialize shutdown of PVS close connection

- **typecheck**
  
  public static boolean typecheck( java.lang.String s )
E.1.3 Class InteractiveTPCPeer

This class hosts a server socket and waits for the TCP to connect. Then, the TPC sends information about the proof status.

E.1.3.1 Declaration

public class InteractiveTPCPeer
extends java.lang.Thread

E.1.3.2 Constructors

- InteractiveTPCPeer
  public InteractiveTPCPeer( jive.PVC.Container.ProofContainer c )

E.1.3.3 Methods

- run
  public void run( )
Appendix F

Package jive.PVC.tactic

<table>
<thead>
<tr>
<th>Package Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td></td>
</tr>
<tr>
<td>Dummy</td>
<td>81</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td></td>
</tr>
<tr>
<td>check_native_call_TACTIC</td>
<td>81</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>eliminate_assumptions_TACTIC</td>
<td>81</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>falsify_TACTIC</td>
<td>82</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>implementation_block_backward_TACTIC</td>
<td>82</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>simple_weak_backward_TACTIC</td>
<td>83</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>swb_ptn_TACTIC</td>
<td>83</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>swb_TACTIC</td>
<td>84</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>Add_Type_Annotation_TACTIC</td>
<td>84</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>SeqEqualAssumptForward_TACTIC</td>
<td>85</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>eliminate_TRUE_TACTIC</td>
<td>85</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>copy_TACTIC</td>
<td>85</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>eliminate_logvar_pre_TACTIC</td>
<td>86</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>equal_assumptions_forward_TACTIC</td>
<td>86</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>inst_assumpt_ptn_TACTIC</td>
<td>87</td>
</tr>
</tbody>
</table>

84
F.1 Interfaces

F.1.1 Interface Dummy

F.1.1.1 Declaration

public interface Dummy

F.2 Classes

F.2.1 Class check_native_call_TACTIC

F.2.1.1 Declaration

public class check_native_call_TACTIC
extends jive.PVC.Container.Tactic

F.2.1.2 Constructors

- check_native_call_TACTIC
  public check_native_call_TACTIC( jive.PVC.Container.ProofContainer c,
  jive.PVC.Container.View.View v )

F.2.1.3 Methods

- call
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )

F.2.2 Class eliminate_assumptions_TACTIC
F.2.2.1 Declaration

```java
public class eliminate_assumptions_TACTIC
    extends jive.PVC.Container.Tactic
```

F.2.2.2 Constructors

- `eliminate_assumptions_TACTIC`
  ```java
  public eliminate_assumptions_TACTIC(
      jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

F.2.2.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )
  ```

F.2.3 Class `falsify_TACTIC`

F.2.3.1 Declaration

```java
public class falsify_TACTIC
    extends jive.PVC.Container.Tactic
```

F.2.3.2 Constructors

- `falsify_TACTIC`
  ```java
  public falsify_TACTIC( jive.PVC.Container.ProofContainer c,
      jive.PVC.Container.View.View v )
  ```

F.2.3.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )
  ```

F.2.4 Class `implementation_block_backward_TACTIC`

F.2.4.1 Declaration

```java
public class implementation_block_backward_TACTIC
    extends jive.PVC.Container.Tactic
```
F.2.4.2 Constructors

- `implementation_block_backward_TACTIC`
  ```java
  public implementation_block_backward_TACTIC(
      jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

F.2.4.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )
  ```

F.2.5 Class `simple_weak_backward_TACTIC`

F.2.5.1 Declaration

```java
public class simple_weak_backward_TACTIC extends jive.PVC.Container.Tactic
```

F.2.5.2 Constructors

- `simple_weak_backward_TACTIC`
  ```java
  public simple_weak_backward_TACTIC(
      jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

F.2.5.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )
  ```

F.2.6 Class `swb_ptn_TACTIC`

F.2.6.1 Declaration

```java
public class swb_ptn_TACTIC extends jive.PVC.Container.Tactic
```

F.2.6.2 Constructors

- `swb_ptn_TACTIC`
  ```java
  public swb_ptn_TACTIC( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```
F.2.6.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )
  ```

F.2.7 Class `swb_TACTIC`

F.2.7.1 Declaration

```java
public class swb_TACTIC
  extends jive.PVC.Container.Tactic
```

F.2.7.2 Constructors

- `swb_TACTIC`
  ```java
  public swb_TACTIC( jive.PVC.Container.ProofContainer c,
           jive.PVC.Container.View.View v )
  ```

F.2.7.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.CompRefString crs,
           jive.PVC.Container.FormulaString post )
  ```

F.2.8 Class `Add_Type_Annotation_TACTIC`

F.2.8.1 Declaration

```java
public class Add_Type_Annotation_TACTIC
  extends jive.PVC.Container.Tactic
```

F.2.8.2 Constructors

- `Add_Type_Annotation_TACTIC`
  ```java
  public Add_Type_Annotation_TACTIC( 
           jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

F.2.8.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )
  ```
F.2.9  Class SeqEqualAssumptForward_TACTIC

F.2.9.1  Declaration

public class SeqEqualAssumptForward_TACTIC
extends jive.PVC.Container.Tactic

F.2.9.2  Constructors

- SeqEqualAssumptForward_TACTIC
  public SeqEqualAssumptForward_TACTIC(
    jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

F.2.9.3  Methods

- call
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn1,
    jive.PVC.Container.ProofTreeNode ptn2 )

F.2.10  Class eliminate_TRUE_TACTIC

F.2.10.1  Declaration

public class eliminate_TRUE_TACTIC
extends jive.PVC.Container.Tactic

F.2.10.2  Constructors

- eliminate_TRUE_TACTIC
  public eliminate_TRUE_TACTIC( jive.PVC.Container.ProofContainer c,
    jive.PVC.Container.View.View v )

F.2.10.3  Methods

- call
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )

F.2.11  Class copy_TACTIC

F.2.11.1  Declaration

public class copy_TACTIC
extends jive.PVC.Container.Tactic
F.2.11.2 Constructors

- *copy*\_TACTIC
  
  public *copy*\_TACTIC( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

F.2.11.3 Methods

- *call*
  
  public TreeNode *call*( jive.PVC.Container.ProofTreeNode ptn )

F.2.12 Class *eliminate*\_logvar\_pre\_TACTIC

F.2.12.1 Declaration

| public class *eliminate*\_logvar\_pre\_TACTIC |
| extends jive.PVC.Container.Tactic |

F.2.12.2 Constructors

- *eliminate*\_logvar\_pre\_TACTIC
  
  public *eliminate*\_logvar\_pre\_TACTIC( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

F.2.12.3 Methods

- *call*
  

F.2.13 Class *equal*\_assumptions\_forward\_TACTIC

F.2.13.1 Declaration

| public class *equal*\_assumptions\_forward\_TACTIC |
| extends jive.PVC.Container.Tactic |

F.2.13.2 Constructors

- *equal*\_assumptions\_forward\_TACTIC
  
  public *equal*\_assumptions\_forward\_TACTIC( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
F.2.13.3 Methods

- `call`
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn1, 
jive.PVC.Container.ProofTreeNode ptn2 )

  - Usage
    * adapts the assumptions of ptn2 to ptn1

F.2.14 Class `inst_assumpt_ptn_TACTIC`

F.2.14.1 Declaration

```java
public class inst_assumpt_ptn_TACTIC 
extends jive.PVC.Container.Tactic
```

F.2.14.2 Constructors

- `inst_assumpt_ptn_TACTIC`
  public inst_assumpt_ptn_TACTIC( jive.PVC.Container.ProofContainer c, 
jive.PVC.Container.View.View v )

F.2.14.3 Methods

- `call`
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )

F.2.15 Class `unroll_subtype_proof_TACTIC`

F.2.15.1 Declaration

```java
public class unroll_subtype_proof_TACTIC 
extends jive.PVC.Container.Tactic
```

F.2.15.2 Constructors

- `unroll_subtype_proof_TACTIC`
  public unroll_subtype_proof_TACTIC( jive.PVC.Container.ProofContainer c, 
jive.PVC.Container.View.View v )
F.2.15.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode g )
  ```

F.2.16 Class `var_forward_TACTIC`

F.2.16.1 Declaration

```java
public class var_forward_TACTIC extends jive.PVC.Container.Tactic
```

F.2.16.2 Constructors

- `var_forward_TACTIC`
  ```java
  public var_forward_TACTIC( jive.PVC.Container.ProofContainer c, 
  jive.PVC.Container.View.View v )
  ```

F.2.16.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn )
  ```

F.2.17 Class `replay_TACTIC`

F.2.17.1 Declaration

```java
public class replay_TACTIC extends jive.PVC.Container.Tactic
```

F.2.17.2 Constructors

- `replay_TACTIC`
  ```java
  public replay_TACTIC( jive.PVC.Container.ProofContainer c, 
  jive.PVC.Container.View.View v )
  ```

F.2.17.3 Methods

- `call`
  ```java
  public ProofTreeNode call( jive.PVC.Container.ProofTreeNode ptn, 
  jive.PVC.Container.FormulaString p, jive.PVC.Container.FormulaString q )
  ```
Appendix G

Package
jive.PVC.Container.Formula

Package Contents

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PVSTokenType</td>
<td>90</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>FormulaParserTokenType</td>
<td>94</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>SubstWalkerTokenType</td>
<td>99</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>UnparseWalkerTokenType</td>
<td>104</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>CollectVarWalkerTokenType</td>
<td>109</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Classes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BooleanFormula</td>
<td>114</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>FormulaSyntaxException</td>
<td>119</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>HashMultiSet</td>
<td>119</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>FormulaLexer</td>
<td>120</td>
</tr>
<tr>
<td>Lexer</td>
<td></td>
</tr>
<tr>
<td>FormulaParser</td>
<td>123</td>
</tr>
<tr>
<td>Parser</td>
<td></td>
</tr>
<tr>
<td>SubstWalker</td>
<td>125</td>
</tr>
<tr>
<td>Walker</td>
<td></td>
</tr>
<tr>
<td>Formula</td>
<td>127</td>
</tr>
<tr>
<td>...no description...</td>
<td></td>
</tr>
<tr>
<td>UnparseWalker</td>
<td>131</td>
</tr>
<tr>
<td>Walker</td>
<td></td>
</tr>
<tr>
<td>CollectVarWalker</td>
<td>132</td>
</tr>
</tbody>
</table>

93
G.1 Interfaces

G.1.1 Interface PVSTokenTypes

G.1.1.1 Declaration

```java
public interface PVSTokenTypes
```

G.1.1.2 Fields

- public static final int EOF
- public static final int NULL_TREE_LOOKAHEAD
- public static final int NOT3
- public static final int FORALL3
- public static final int EXISTS3
- public static final int IFF3
- public static final int IMPLIES3
- public static final int WHEN3
- public static final int OR3
- public static final int XOR3
- public static final int ORELSE3
- public static final int AND3
  -
- public static final int ANDTHEN3
  -
- public static final int IF3
  -
- public static final int O3
  -
- public static final int THEN3
  -
- public static final int ELSE3
  -
- public static final int ENDIF3
  -
- public static final int TILDE
  -
- public static final int TURNS
  -
- public static final int DTURNS
  -
- public static final int IFF
  -
- public static final int IMPLIES
  -
- public static final int OR
  -
- public static final int AND
  -
- public static final int AND2
  -
• public static final int EQU
  -
• public static final int NEQ
  -
• public static final int CEQ
  -
• public static final int LESS
  -
• public static final int LEQ
  -
• public static final int GRE
  -
• public static final int GEQ
  -
• public static final int DLESS
  -
• public static final int DGRE
  -
• public static final int DLEQ
  -
• public static final int DGEQ
  -
• public static final int LLESS
  -
• public static final int LGRE
  -
• public static final int AT
  -
• public static final int CROSS
  -
• public static final int DAT
  -
- Public static final int DCROSS
- Public static final int PLUS
- Public static final int MINUS
- Public static final int DPLUS
- Public static final int TIMES
- Public static final int DIV
- Public static final int DTIMES
- Public static final int DDIV
- Public static final int SQUARE
- Public static final int RHOMB
- Public static final int CARET
- Public static final int DCARET
- Public static final int LPAR
- Public static final int RPAR
- Public static final int COMMA
- Public static final int DOT
- public static final int COLON
  -
- public static final int LSB
  -
- public static final int RSB
  -
- public static final int LFB
  -
- public static final int RFB
  -
- public static final int LDB
  -
- public static final int RDB
  -
- public static final int LTB
  -
- public static final int RTB
  -
- public static final int ASSIGN
  -
- public static final int ASSIGN2
  -
- public static final int ID
  -
- public static final int STRING
  -
- public static final int NUMBER
  -
- public static final int WS
  -

G.1.2 Interface FormulaParserTokenType
G.1.2.1 Declaration

```
public interface FormulaParserTokenTypes
```

G.1.2.2 Fields

- public static final int EOF
- public static final int NULL_TREE_LOOKAHEAD
- public static final int NOT3
- public static final int FORALL3
- public static final int EXISTS3
- public static final int IFF3
- public static final int IMPLIES3
- public static final int WHEN3
- public static final int OR3
- public static final int XOR3
- public static final int ORELSE3
- public static final int AND3
- public static final int ANDTHEN3
- public static final int IF3
• public static final int O3

• public static final int THEN3

• public static final int ELSE3

• public static final int ENDIF3

• public static final int TILDE

• public static final int TURNS

• public static final int DTURNS

• public static final int IFF

• public static final int IMPLIES

• public static final int OR

• public static final int AND

• public static final int AND2

• public static final int EQU

• public static final int NEQ

• public static final int CEQ
- public static final int LESS
  -
- public static final int LEQ
  -
- public static final int GRE
  -
- public static final int GEQ
  -
- public static final int DLESS
  -
- public static final int DGRE
  -
- public static final int DLEQ
  -
- public static final int DGEQ
  -
- public static final int LLESS
  -
- public static final int LGRE
  -
- public static final int AT
  -
- public static final int CROSS
  -
- public static final int DAT
  -
- public static final int DCROSS
  -
- public static final int PLUS
  -
- public static final int MINUS
  -
• public static final int DPLUS
  -
• public static final int TIMES
  -
• public static final int DIV
  -
• public static final int DTIMES
  -
• public static final int DDIV
  -
• public static final int SQUARE
  -
• public static final int RHOMB
  -
• public static final int CARET
  -
• public static final int DCARET
  -
• public static final int LPAR
  -
• public static final int RPAR
  -
• public static final int COMMA
  -
• public static final int DOT
  -
• public static final int COLON
  -
• public static final int LSB
  -
• public static final int RSB
  -
• public static final int LFB
  
• public static final int RFB
  
• public static final int LDB
  
• public static final int RDB
  
• public static final int LTB
  
• public static final int RTB
  
• public static final int ASSIGN
  
• public static final int ASSIGN2
  
• public static final int ID
  
• public static final int STRING
  
• public static final int NUMBER
  
• public static final int WS

G.1.3 Interface SubstWalkerTokenType

G.1.3.1 Declaration

```java
public interface SubstWalkerTokenType
```
G.1.3.2 Fields

- public static final int EOF

- public static final int NULL_TREE_LOOKAHEAD

- public static final int NOT3

- public static final int FORALL3

- public static final int EXISTS3

- public static final int IFF3

- public static final int IMPLIES3

- public static final int WHEN3

- public static final int OR3

- public static final int XOR3

- public static final int ORELSE3

- public static final int AND3

- public static final int ANDTHEN3

- public static final int IF3

- public static final int O3
- public static final int THEN3
  
- public static final int ELSE3
  
- public static final int ENDIF3
  
- public static final int TILDE
  
- public static final int TURNS
  
- public static final int DTURNS
  
- public static final int IFF
  
- public static final int IMPLIES
  
- public static final int OR
  
- public static final int AND
  
- public static final int AND2
  
- public static final int EQU
  
- public static final int NEQ
  
- public static final int CEQ
  
- public static final int LESS
  
- public static final int LEQ
• public static final int GRE
  
• public static final int GEQ
  
• public static final int DLESS
  
• public static final int DGRE
  
• public static final int DLEQ
  
• public static final int DGEQ
  
• public static final int LLESS
  
• public static final int LGRE
  
• public static final int AT
  
• public static final int CROSS
  
• public static final int DAT
  
• public static final int DCROSS
  
• public static final int PLUS
  
• public static final int MINUS
  
• public static final int DPLUS
  
• public static final int TIMES
• public static final int DIV
  -
• public static final int DTIMES
  -
• public static final int DDIV
  -
• public static final int SQUARE
  -
• public static final int RHOMB
  -
• public static final int CARET
  -
• public static final int DCARET
  -
• public static final int LPAR
  -
• public static final int RPAR
  -
• public static final int COMMA
  -
• public static final int DOT
  -
• public static final int COLON
  -
• public static final int LSB
  -
• public static final int RSB
  -
• public static final int LFB
  -
• public static final int RFB
  -
• public static final int LDB
  –
• public static final int RDB
  –
• public static final int LTB
  –
• public static final int RTB
  –
• public static final int ASSIGN
  –
• public static final int ASSIGN2
  –
• public static final int ID
  –
• public static final int STRING
  –
• public static final int NUMBER
  –
• public static final int WS
  –

G.1.4 Interface UnparseWalkerTokenType

G.1.4.1 Declaration

```java
public interface UnparseWalkerTokenType
```

G.1.4.2 Fields

• public static final int EOF
  –
• public static final int NULL_TREE_LOOKAHEAD
  –
• public static final int NOT3
- public static final int FORALL3
- public static final int EXISTS3
- public static final int IFF3
- public static final int IMPLIES3
- public static final int WHEN3
- public static final int OR3
- public static final int XOR3
- public static final int ORELSE3
- public static final int AND3
- public static final int ANDTHEN3
- public static final int IF3
- public static final int O3
- public static final int THEN3
- public static final int ELSE3
- public static final int ENDIF3
• public static final int TILDE
  -
• public static final int TURNS
  -
• public static final int DTURNS
  -
• public static final int IFF
  -
• public static final int IMPLIES
  -
• public static final int OR
  -
• public static final int AND
  -
• public static final int AND2
  -
• public static final int EQU
  -
• public static final int NEQ
  -
• public static final int CEQ
  -
• public static final int LESS
  -
• public static final int LEQ
  -
• public static final int GRE
  -
• public static final int GEQ
  -
• public static final int DLESS
  -
• public static final int DGRE
  -
• public static final int DLEQ
  -
• public static final int DGEQ
  -
• public static final int LLESS
  -
• public static final int LGRE
  -
• public static final int AT
  -
• public static final int CROSS
  -
• public static final int DAT
  -
• public static final int DCROSS
  -
• public static final int PLUS
  -
• public static final int MINUS
  -
• public static final int DPLUS
  -
• public static final int TIMES
  -
• public static final int DIV
  -
• public static final int DTIMES
  -
• public static final int DDIV
  -
• public static final int SQUARE
  –
• public static final int RHOMB
  –
• public static final int CARET
  –
• public static final int DCARET
  –
• public static final int LPAR
  –
• public static final int RPAR
  –
• public static final int COMMA
  –
• public static final int DOT
  –
• public static final int COLON
  –
• public static final int LSB
  –
• public static final int RSB
  –
• public static final int LFB
  –
• public static final int RFB
  –
• public static final int LDB
  –
• public static final int RDB
  –
• public static final int LTB
  –
- public static final int RTB
  -
- public static final int ASSIGN
  -
- public static final int ASSIGN2
  -
- public static final int ID
  -
- public static final int STRING
  -
- public static final int NUMBER
  -
- public static final int WS
  -

G.1.5  *Interface CollectVarWalkerTokenType*

G.1.5.1 Declaration

```
public interface CollectVarWalkerTokenType
```

G.1.5.2 Fields

- public static final int EOF
  -
- public static final int NULL_TREE_LOOKAHEAD
  -
- public static final int NOT3
  -
- public static final int FORALL3
  -
- public static final int EXISTS3
  -
- public static final int IFF3
• public static final int IMPLIES3

• public static final int WHEN3

• public static final int OR3

• public static final int XOR3

• public static final int ORElse3

• public static final int AND3

• public static final int ANDThen3

• public static final int IF3

• public static final int O3

• public static final int THEN3

• public static final int ELSE3

• public static final int ENDIF3

• public static final int TILDE

• public static final int TURNS

• public static final int DTURNS
• public static final int IFF
  
• public static final int IMPLIES
  
• public static final int OR
  
• public static final int AND
  
• public static final int AND2
  
• public static final int EQU
  
• public static final int NEQ
  
• public static final int CEQ
  
• public static final int LESS
  
• public static final int LEQ
  
• public static final int GRE
  
• public static final int GEQ
  
• public static final int DLESS
  
• public static final int DGRE
  
• public static final int DLEQ
  
• public static final int DGEQ
• public static final int LLESS
  -
• public static final int LGRE
  -
• public static final int AT
  -
• public static final int CROSS
  -
• public static final int DAT
  -
• public static final int DCROSS
  -
• public static final int PLUS
  -
• public static final int MINUS
  -
• public static final int DPLUS
  -
• public static final int TIMES
  -
• public static final int DIV
  -
• public static final int DTIMES
  -
• public static final int DDIV
  -
• public static final int SQUARE
  -
• public static final int RHOMB
  -
• public static final int CARET
  -
• public static final int DCARET
  
• public static final int LPAR
  
• public static final int RPAR
  
• public static final int COMMA
  
• public static final int DOT
  
• public static final int COLON
  
• public static final int LSB
  
• public static final int RSB
  
• public static final int LFB
  
• public static final int RFB
  
• public static final int LDB
  
• public static final int RDB
  
• public static final int LTB
  
• public static final int RTB
  
• public static final int ASSIGN
  
• public static final int ASSIGN2
• public static final int ID
  –
• public static final int STRING
  –
• public static final int NUMBER
  –
• public static final int WS
  –

G.2 Classes

G.2.1 Class BooleanFormula

G.2.1.1 Declaration

```java
public class BooleanFormula
  extends jive.PVC.Container.Formula.Formula
```

G.2.1.2 Fields

• public static final BooleanFormula TRUE
  –
• public static final BooleanFormula FALSE
  –

G.2.1.3 Constructors

• BooleanFormula
  protected BooleanFormula( antlr.collections.AST a )
    – Usage
      * build a new BooleanFormula from an existing AST we assume that a ist already typechecked

• BooleanFormula
  public BooleanFormula( java.lang.String s )
    – Usage
* builds a new BooleanFormula s is not typechecked syntax errors are not handled use carefully! (supposed to be used for automatically generated formulas only)

- **BooleanFormula**
  
  public BooleanFormula( java.lang.String s, jive.PC.Program.ComRef ref, java.util.Hashtable st )

  - **Usage**
    * Constructor for a boolean formula parses and typechecks s s must be valid in ref

- **BooleanFormula**
  
  public BooleanFormula( java.lang.String s, java.util.Hashtable st )

  - **Usage**
    * Constructor for boolean formula parses and typechecks s s must be a sigma-formula

### G.2.1.4 Methods

- **and**
  
  public BooleanFormula and( jive.PVC.Container.Formula.BooleanFormula f )

- **andold**
  
  public BooleanFormula andold( jive.PVC.Container.Formula.BooleanFormula f )

  - **Usage**
    * returns new Formula representing this AND f

- **checkConjunction**
  
  public void checkConjunction( )

  - **Usage**
    * helper method. checks if a BooleanFormula is a conjunction

- **checkDisjunction**
  
  public void checkDisjunction( )

  - **Usage**
    * helper method. checks if a BooleanFormula is a disjunction

- **checkFormulaEqual**
  
  public void checkFormulaEqual( jive.PVC.Container.Formula.BooleanFormula b, java.lang.String message )
Usage
  * helper method, checks if two formulae are equal, if not throws ReqException with text string

- `checkGammaPost`
  public void `checkGammaPost( jive.PC.Program.MethodRef mr )`

  Usage
  * checks if a formula f is a gamma-post-formula

- `checkGammaPre`
  public void `checkGammaPre( jive.PC.Program.MethodRef mr )`

  Usage
  * checks if a formula f is a gamma-pre-formula

- `checkIsConjunct`
  public void `checkIsConjunct( jive.PVC.Container.Formula.BooleanFormula g )`

  Usage
  * helper method, checks if g is a conjunct of f

- `copy`
  public Formula `copy( )`

  Usage
  * get a copy of this Formula

- `exists`
  public BooleanFormula `exists( java.lang.String v, java.lang.String sort )`

  Usage
  * returns new Formula representing EXISTS (v: sort): this

- `forall`
  public BooleanFormula `forall( java.lang.String v, java.lang.String sort )`

  Usage
  * returns new Formula representing FORALL (v: sort): this

- `getLemma`
  public String `getLemma( jive.PC.Program.CompRef ref, java.util.Hashtable st )`

- `getOperand1`
  public BooleanFormula `getOperand1( )`
Usage
* returns first operand of binary expression we expect that root is binary
  boolean operator otherwise, the result is undefined

g_operand2
public BooleanFormula getOperand2( )

Usage
* returns second operand of binary expression we expect that root is binary
  boolean operator otherwise, the result is undefined

implies
public BooleanFormula implies( jive.PVC.Container.Formula.BooleanFormula f )

Usage
* returns new Formula representing this IMPLIES f

isConjunct
public boolean isConjunct( jive.PVC.Container.Formula.BooleanFormula f )

Usage
* test if f is a top level conjunct of this E.g.: this = (A /“B) /“(f /“C)

main
public static void main( java.lang.String [] argv )

not
public BooleanFormula not( )

Usage
* returns new Formula representing NOT (this)

or
public BooleanFormula or( jive.PVC.Container.Formula.BooleanFormula f )

Usage
* returns new Formula representing this OR f

read
public static BooleanFormula read( java.io.BufferedReader br )

Usage
* reads a BooleanFormula from the given BufferedReader. The formula is
  assumed to be correct
• removeConjunct
  public BooleanFormula removeConjunct(
    jive.PVC.Container.Formula.BooleanFormula f )
  
    Usage
    * remove f, if f is a top level conjunct of this E.g.: this = (A / “B) /“(f / “C)
      ->this = (A / “B) /“(true / "C")

• removeConjunctAST
  protected static AST removeConjunctAST( antlr.collections.AST tree, 
    antlr.collections.AST c )

• substitute
  public BooleanFormula substitute( java.util.HashTable sl,
    jive.PC.Program.CompRef fromRef, jive.PC.Program.CompRef toRef, 
    java.util.Hashtable symboltable )
  
    Usage
    * substitute all pairs contained in sl substitutes value for key in sl

• substitute
  public BooleanFormula substitute( java.util.HashTable sl,
    jive.PC.Program.CompRef ref, java.util.Hashtable symboltable )

• substitute
  public BooleanFormula substitute( java.lang.String [] from,
    java.lang.String [] to, jive.PC.Program.MethodRef crfrom,
    jive.PC.Program.MethodRef crito, java.util.Hashtable symboltable )

• substitute
  public BooleanFormula substitute( java.lang.String from,
    java.lang.String to, jive.PC.Program.CompRef fromRef,
    jive.PC.Program.CompRef toRef, java.util.Hashtable symboltable )
  
    Usage
    * substitute to for from in this

• substitute
  public BooleanFormula substitute( java.lang.String from,
    java.lang.String to, jive.PC.Program.CompRef ref, java.util.Hashtable 
    symboltable )

• unbindVariable
  public BooleanFormula unbindVariable( java.lang.String v )
  
    Usage
    * this has to be a quantified formula remove v from the list of bound variables
      remove quantifier if v is the only bound variable of this quantifier
• write
  public void write( java.io.PrintWriter sw )
  
  - Usage
    * writes this BooleanFormula into the PrintWriter

G.2.2 Class FormulaSyntaxException

G.2.2.1 Declaration

public class FormulaSyntaxException extends java.lang.Exception

G.2.2.2 Constructors

• FormulaSyntaxException
  public FormulaSyntaxException( java.lang.String s )

G.2.3 Class HashMultiSet

G.2.3.1 Declaration

public class HashMultiSet extends java.util.Hashtable

G.2.3.2 Constructors

• HashMultiSet
  public HashMultiSet( )

G.2.3.3 Methods

• addAll
  public void addAll( java.util.Vector ms )
• put
  public Object put( java.lang.Object key, java.lang.Object value )
• remove
  public Object remove( java.lang.Object key )
• removeAll
  public void removeAll( java.util.Vector ms )
G.2.4 Class FormulaLexer

 Lexer

 G.2.4.1 Declaration

```java
public class FormulaLexer
  extends antlr.CharScanner
  implements PVSTokenTypes, antlr.TokenStream
```

 G.2.4.2 Fields

- public static final BitSet _tokenSet_0

 G.2.4.3 Constructors

- `FormulaLexer`
  ```java
  public FormulaLexer( antlr.InputBuffer ib )
  ```

- `FormulaLexer`
  ```java
  public FormulaLexer( java.io.InputStream in )
  ```

- `FormulaLexer`
  ```java
  public FormulaLexer( antlr.LexerSharedInputState state )
  ```

- `FormulaLexer`
  ```java
  public FormulaLexer( java.io.Reader in )
  ```

 G.2.4.4 Methods

- `mAND`
  ```java
  public final void mAND( boolean _createToken )
  ```

- `mAND2`
  ```java
  public final void mAND2( boolean _createToken )
  ```

- `mASSIGN`
  ```java
  public final void mASSIGN( boolean _createToken )
  ```

- `mASSIGN2`
  ```java
  public final void mASSIGN2( boolean _createToken )
  ```

- `mAT`
  ```java
  public final void mAT( boolean _createToken )
  ```
• \texttt{mCARET}
  \hspace{1cm} \texttt{public final void mCARET( boolean _createToken )}

• \texttt{mCEQ}
  \hspace{1cm} \texttt{public final void mCEQ( boolean _createToken )}

• \texttt{mCOLON}
  \hspace{1cm} \texttt{public final void mCOLON( boolean _createToken )}

• \texttt{mCOMMA}
  \hspace{1cm} \texttt{public final void mCOMMA( boolean _createToken )}

• \texttt{mCROSS}
  \hspace{1cm} \texttt{public final void mCROSS( boolean _createToken )}

• \texttt{mDAT}
  \hspace{1cm} \texttt{public final void mDAT( boolean _createToken )}

• \texttt{mDCARET}
  \hspace{1cm} \texttt{public final void mDCARET( boolean _createToken )}

• \texttt{mDCROSS}
  \hspace{1cm} \texttt{public final void mDCROSS( boolean _createToken )}

• \texttt{mDDIV}
  \hspace{1cm} \texttt{public final void mDDIV( boolean _createToken )}

• \texttt{mDGEQ}
  \hspace{1cm} \texttt{public final void mDGEQ( boolean _createToken )}

• \texttt{mDGRE}
  \hspace{1cm} \texttt{public final void mDGRE( boolean _createToken )}

• \texttt{mDIV}
  \hspace{1cm} \texttt{public final void mDIV( boolean _createToken )}

• \texttt{mDLEQ}
  \hspace{1cm} \texttt{public final void mDLEQ( boolean _createToken )}

• \texttt{mDLESS}
  \hspace{1cm} \texttt{public final void mDLESS( boolean _createToken )}

• \texttt{mDOT}
  \hspace{1cm} \texttt{public final void mDOT( boolean _createToken )}

• \texttt{mDPLUS}
  \hspace{1cm} \texttt{public final void mDPLUS( boolean _createToken )}

• \texttt{mDTIMES}
  \hspace{1cm} \texttt{public final void mDTIMES( boolean _createToken )}

• \texttt{mDTURNS}
  \hspace{1cm} \texttt{public final void mDTURNS( boolean _createToken )}
- `mEQU`
  ```java
  public final void mEQU( boolean _createToken )
  ```

- `mGEQ`
  ```java
  public final void mGEQ( boolean _createToken )
  ```

- `mGRE`
  ```java
  public final void mGRE( boolean _createToken )
  ```

- `mID`
  ```java
  public final void mID( boolean _createToken )
  ```

- `mIFF`
  ```java
  public final void mIFF( boolean _createToken )
  ```

- `mIMPLIES`
  ```java
  public final void mIMPLIES( boolean _createToken )
  ```

- `mLDB`
  ```java
  public final void mLDB( boolean _createToken )
  ```

- `mLEQ`
  ```java
  public final void mLEQ( boolean _createToken )
  ```

- `mLESS`
  ```java
  public final void mLESS( boolean _createToken )
  ```

- `mLFB`
  ```java
  public final void mLFB( boolean _createToken )
  ```

- `mLGRE`
  ```java
  public final void mLGRE( boolean _createToken )
  ```

- `mLLESS`
  ```java
  public final void mLLESS( boolean _createToken )
  ```

- `mLPAR`
  ```java
  public final void mLPAR( boolean _createToken )
  ```

- `mLSB`
  ```java
  public final void mLSB( boolean _createToken )
  ```

- `mLTB`
  ```java
  public final void mLTB( boolean _createToken )
  ```

- `mMINUS`
  ```java
  public final void mMINUS( boolean _createToken )
  ```

- `mNEQ`
  ```java
  public final void mNEQ( boolean _createToken )
  ```

- `mNUMBER`
  ```java
  public final void mNUMBER( boolean _createToken )
  ```
- mOR
  public final void mOR( boolean _createToken )

- mPLUS
  public final void mPLUS( boolean _createToken )

- mRDB
  public final void mRDB( boolean _createToken )

- mRFB
  public final void mRFB( boolean _createToken )

- mRHOMB
  public final void mRHOMB( boolean _createToken )

- mRPAR
  public final void mRPAR( boolean _createToken )

- mRSB
  public final void mRSB( boolean _createToken )

- mRTB
  public final void mRTB( boolean _createToken )

- mSQUARE
  public final void mSQUARE( boolean _createToken )

- mSTRING
  public final void mSTRING( boolean _createToken )

- mTILDE
  public final void mTILDE( boolean _createToken )

- mTIMES
  public final void mTIMES( boolean _createToken )

- mTURNS
  public final void mTURNS( boolean _createToken )

- mWS
  public final void mWS( boolean _createToken )

- nextToken
  public Token nextToken( )

G.2.5 Class FormulaParser

Parser

G.2.5.1 Declaration

public class FormulaParser
  extends antlr.LLkParser
  implements FormulaParserTokenType
G.2.5.2 Fields

- public static final String tokenNames

G.2.5.3 Constructors

- FormulaParser
  public FormulaParser( antlr.ParserSharedInputState state )

- FormulaParser
  public FormulaParser( antlr.TokenBuffer tokenBuf )

- FormulaParser
  protected FormulaParser( antlr.TokenBuffer tokenBuf, int k )

- FormulaParser
  public FormulaParser( antlr.TokenStream lexer )

- FormulaParser
  protected FormulaParser( antlr.TokenStream lexer, int k )

G.2.5.4 Methods

- args
  public final void args( )

- arguments
  public final void arguments( )

- bindings
  public final void bindings( )

- expr
  public final void expr( )

- expr0
  public final void expr0( )

- expr00
  public final void expr00( )

- expr1
  public final void expr1( )

- expr10
  public final void expr10( )
• `expr11`
  public final void `expr11()`
• `expr12`
  public final void `expr12()`
• `expr13`
  public final void `expr13()`
• `expr14`
  public final void `expr14()`
• `expr15`
  public final void `expr15()`
• `expr16`
  public final void `expr16()`
• `expr2`
  public final void `expr2()`
• `expr3`
  public final void `expr3()`
• `expr4`
  public final void `expr4()`
• `expr5`
  public final void `expr5()`
• `expr6`
  public final void `expr6()`
• `expr9`
  public final void `expr9()`
• `moduleActuals`
  public final void `moduleActuals()`
• `primary`
  public final void `primary()`
• `typedid`
  public final void `typedid()`
• `typedids`
  public final void `typedids()`

G.2.6  Class SubstWalker

Walker
G.2.6.1 Declaration

```java
public class SubstWalker
extends antlr.TreeParser
implements SubstWalkerTokenTypes
```

G.2.6.2 Fields

- public static final String _tokenNames
  -
- public static final BitSet _tokenSet_0
  -

G.2.6.3 Constructors

- `SubstWalker`
  ```java
  public SubstWalker()
  ```

G.2.6.4 Methods

- `addBoundVars`
  ```java
  public void addBoundVars( java.util.Vector v )
  ```

- `arguments`
  ```java
  public final boolean arguments( antlr.collections.AST _t )
  ```

- `bindings`
  ```java
  public final Vector bindings( antlr.collections.AST _t )
  ```

- `expr`
  ```java
  public final void expr( antlr.collections.AST _t )
  ```

- `isFreeVar`
  ```java
  public boolean isFreeVar( java.lang.String s )
  ```

- `moduleActuals`
  ```java
  public final boolean moduleActuals( antlr.collections.AST _t )
  ```

- `removeBoundVars`
  ```java
  public void removeBoundVars( java.util.Vector v )
  ```

- `setSubstTable`
  ```java
  public void setSubstTable( java.util.Hashtable st )
  ```

- `typedid`
  ```java
  public final String typedid( antlr.collections.AST _t )
  ```
G.2.7 Class Formula

G.2.7.1 Declaration

```java
public class Formula
extends java.lang.Object
```

G.2.7.2 Constructors

- `Formula`
  protected `Formula()`
  - Usage
    * default constructor needed for BooleanFormula only

- `Formula`
  protected `Formula(antlr.collections.AST a)`
  - Usage
    * build a new Formula from an existing AST we assume that a is already typechecked

- `Formula`
  protected `Formula(jive.PVC.Container.Formula.Formula f)`
  - Usage
    * copy constructor

- `Formula`
  public `Formula(java.lang.String s, jive.PC.Program.ComprRef ref, java.lang.String sort, java.util.Hashtable st)`
  - Usage
    * Constructor for Formula parses and typechecks s s must be valid in ref

- `Formula`
  public `Formula(java.lang.String s, java.lang.String t,
  java.util.Hashtable st)`
  - Usage
    * Constructor for Formula parses and typechecks s s must be a sigma-formula
G.2.7.3 Methods

- **checkGammaPos**
  ```java
class checkGammaPos {
  public void checkGammaPos( jive.PC.Program.MethodRef mr ) {
    Usage
    * checks if a formula f is a gamma-post-formula
  }
}
```

- **checkGammaPre**
  ```java
class checkGammaPre {
  public void checkGammaPre( jive.PC.Program.MethodRef mr ) {
    Usage
    * checks if a formula f is a gamma-pre-formula
  }
}
```

- **checkSigma**
  ```java
class checkSigma {
  public void checkSigma( jive.PC.Program.ComRef cr ) {
    Usage
    * checks whether a formula f is a sigma-formula
  }
}
```

- **checkSigma**
  ```java
class checkSigma {
  public static void checkSigma( java.lang.String s,
                              jive.PC.Program.ComRef cr,
                              java.lang.String sort,
                              java.util.Hashtable h ) {
  }
}
```

- **close**
  ```java
class close {
  protected String close( java.lang.String t,
                         jive.PC.Program.ComRef ref,
                         java.util.Hashtable st ) {
    Usage
    * build the closed form of a formula by binding all free variables in context ref
    returns a String (as opposed to Formula) because closed forms will usually be
    passed directly to PVS
    ref can be null for Sigma formulas
  }
}
```

- **containsVar**
  ```java
class containsVar {
  protected boolean containsVar( antlr.collections.AST binding,
                                java.lang.String v ) {
    Usage
    * checks whether v is contained in a list of bindings binding can be null
  }
}
```

- **containsVariable**
  ```java
class containsVariable {
  public boolean containsVariable( java.lang.String s ) {
    Usage
    * check whether s is a free variable in this
  }
}
```
• *copy*
  public Formula copy( )

  – Usage
  * get a copy of this Formula

• *copyAST*
  protected static AST copyAST( antlr.collections.AST a )

  – Usage
  * copy whole AST

• *equals*
  public boolean equals( jive.PVC.Container.Formula.Formula f )

  – Usage
  * compare two formulas alternative syntax (e.g, AND, &, and /““) is regarded as equal

• *equalsAST*
  protected static boolean equalsAST( antlr.collections.AST a1, antlr.collections.AST a2 )

  – Usage
  * compares two AST's alternative syntax (e.g, AND, &, and /““) is regarded as equal

• *freeVars*
  public Vector freeVars( )

  – Usage
  * collect all free variables of formula

• *getFString*
  public FormulaString getFString( )

  – Usage
  * returns this Formula as a FormulaString

• *getPVS Lemma*
  public String getPVS Lemma( jive.PC.Program.ComprRef ref, java.util.Hashtable st )

• *getVariableBinding*
  protected String getVariableBinding( jive.PC.Program.ComprRef ref, java.util.Hashtable st )

• *isConjunct*
  protected static boolean isConjunct( antlr.collections.AST t, antlr.collections.AST f )
- **Usage**  
  * test if f is a top level conjunct of t E.g.: \( t = (A \lor B) \lor (f \lor C) \) true if t equals f

- **isConjunction**  
  public boolean isConjunction( )  
  - **Usage**  
    * checks whether formula is a conjunction

- **isConjunction**  
  protected static boolean isConjunction( antlr.collections.AST a )  
  - **Usage**  
    * check whether a is a conjunction

- **isDisjunction**  
  public boolean isDisjunction( )  
  - **Usage**  
    * checks whether formula is a disjunction

- **isExQuantified**  
  public boolean isExQuantified( java.lang.String v )  
  - **Usage**  
    * checks whether v is existentially quantified on the outmost level

- **isGammaPost**  
  public boolean isGammaPost( jive.PC.Program.MethodRef ref )  
  - **Usage**  
    * checks (by consulting PC) whether formula is a gamma post formula

- **isGammaPre**  
  public boolean isGammaPre( jive.PC.Program.MethodRef ref )  
  - **Usage**  
    * checks (by consulting PC) whether formula is a gamma pre formula

- **isImplication**  
  public boolean isImplication( )  
  - **Usage**  
    * checks whether formula is a implication

- **isQuantified**  
  protected boolean isQuantified( java.lang.String v, antlr.collections.AST f, int q )
- **Usage**
  * checks whether \( v \) is quantified in \( f \) with quantor \( q \)

- **is Sigma**
  public boolean isSigma( jive.PC.Program.ComprRef ref )

  - **Usage**
    * checks (by consulting PC) whether formula is a sigma formula note that
    formula may contain PVS constants (e.g., true) which are neither declared in
    the symbol table nor program variables. Thus, we cannot use the symbol
table to implement isSigma.

- **is Type Name**
  public boolean isTypeName( )

  - **Usage**
    * check whether formula is a valid type name

- **is Uni Quantified**
  public boolean isUniQuantified( java.lang.String v )

  - **Usage**
    * checks whether \( v \) is universally quantified on the outmost level

- **main**
  public static void main( java.lang.String [] argv )

- **parse**
  protected AST parse( java.lang.String s )

  - **Usage**
    * parse \( s \) and return abstract syntax tree

- **toString**
  public String toString( )

  - **Usage**
    * returns text representation in PVS syntax

- **typecheck**
  protected void typecheck( java.lang.String t, jive.PC.Program.ComprRef
ref, java.util.Hashtable st )

  - **Usage**
    * typecheck Formula by sending and typechecking lemma in TPC \( t \): sort of
    formula

**G.2.8 Class Unparse Walker**

Walker
G.2.8.1 Declaration

```java
public class UnparseWalker
extends antlr.TreeParser
implements UnparseWalker.TokenTypes
```

G.2.8.2 Fields

- public static final String _tokenNames
  -
- public static final BitSet _tokenSet_0
  -

G.2.8.3 Constructors

- `UnparseWalker`
  - public `UnparseWalker()`
- `UnparseWalker`
  - public `UnparseWalker(boolean pv)`

G.2.8.4 Methods

- `arguments`
  - public final String `arguments(antlr.collections.AST t)`
- `bindings`
  - public final String `bindings(antlr.collections.AST t)`
- `expr`
  - public final String `expr(antlr.collections.AST t)`
- `moduleActuals`
  - public final String `moduleActuals(antlr.collections.AST t)`
- `par`
  - public String `par(java.lang.String s)`
- `typeid`
  - public final String `typeid(antlr.collections.AST t)`

G.2.9 Class CollectVarWalker

Walker
G.2.9.1 Declaration

```
public class CollectVarWalker
extends antlr.TreeParser
implements CollectVarWalkerTokenTypes
```

G.2.9.2 Fields

- public static final String _tokenNames
- public static final BitSet _tokenSet_0

G.2.9.3 Constructors

- CollectVarWalker
  public CollectVarWalker()

G.2.9.4 Methods

- arguments
  public final Vector arguments( antlr.collections.AST _t )
- bindings
  public final Vector bindings( antlr.collections.AST _t )
- expr
  public final Vector expr( antlr.collections.AST _t )
- merge
  public Vector merge( java.util.Vector a, java.util.Vector b )
  - Usage
    * merge b into a, do not copy elements that are already in a
- moduleActuals
  public final Vector moduleActuals( antlr.collections.AST _t )
- typedid
  public final String typedid( antlr.collections.AST _t )
## Appendix H

### Package jive.PVC.Container

**Package Contents**

<table>
<thead>
<tr>
<th>Classes</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumptions</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>CompRefString</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>FormulaString</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>LogicalVarString</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>Rule</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>Tactic</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>ProgVarString</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>subst_backward</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>call_backward</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>ReqException</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>This Exception is thrown, iff a requirement test fails.</td>
</tr>
<tr>
<td>all_backward</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>all_forward</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>assumpt_elim_backward</td>
<td>143</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>assumpt_elim_forward</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>ContextInfo</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td>...no description...</td>
</tr>
<tr>
<td>assumpt_intro_backward</td>
<td>145</td>
</tr>
</tbody>
</table>
...no description...
assumpt_intro_forward ................................................. 145
...no description...
block_backward .......................................................... 145
...no description...
block_forward .......................................................... 146
...no description...
call_forward ........................................................... 146
...no description...
var_backward ............................................................ 147
...no description...
Triple ................................................................. 147
...no description...
check_assign_axiom .................................................... 149
...no description...
check_assumpt_axiom ................................................... 149
...no description...
check_false_axiom ...................................................... 149
...no description...
check_field_read_axiom .............................................. 150
...no description...
check_field_write_axiom ............................................. 150
...no description...
check_return_axiom ................................................... 151
...no description...
class_backward ......................................................... 151
...no description...
class_forward ........................................................ 152
...no description...
conjunct_backward .................................................... 152
...no description...
conjunct_forward ...................................................... 152
...no description...
disjunct_backward ..................................................... 153
...no description...
disjunct_forward ....................................................... 153
...no description...
ex_backward .......................................................... 154
...no description...
ex_forward .......................................................... 154
...no description...
if_backward ............................................................. 155
...no description...
if_forward ............................................................. 155
...no description...
implementation_backward ........................................... 155
...no description...
implementation_forward .................................................. 156
  ...no description..
inst_assign_axiom .......................................................... 156
  ...no description..
inst_assumpt_axiom .......................................................... 157
  ...no description..
inst_false_axiom ............................................................. 157
  ...no description..
inst_field_read_axiom ...................................................... 158
  ...no description..
inst_field_write_axiom ..................................................... 158
  ...no description..
inst_return_axiom ........................................................... 158
  ...no description..
inv_backward ................................................................. 159
  ...no description..
inv_forward ................................................................. 159
  ...no description..
var_forward ................................................................. 160
  ...no description..
invocation_backward ....................................................... 160
  ...no description..
invocation_forward ........................................................ 161
  ...no description..
seq_back_backward .......................................................... 161
  ...no description..
seq_front_backward ........................................................ 161
  ...no description..
seq_forward ................................................................. 162
  ...no description..
static_invocation_backward ............................................... 162
  ...no description..
static_invocation_forward ............................................... 163
  ...no description..
strength_backward ........................................................ 163
  ...no description..
check_cast_axiom ........................................................... 164
  ...no description..
check_empty_axiom ........................................................... 164
  ...no description..
subst_forward ............................................................... 164
  ...no description..
strength_forward ........................................................ 165
  ...no description..
subtype_backward .......................................................... 165
  ...no description..
subtype_forward ............................................................ 166
...no description...
weak_backward..........................................................166
...no description...
weak_forward............................................................167
...no description...
while_backward..........................................................167
...no description...
while_forward............................................................167
...no description...
Sequent.................................................................168
...no description...
inst_cast_axiom..........................................................170
...no description...
inst_empty_axiom..........................................................170
...no description...
ProofTreeNode............................................................171
...no description...
ProofContainer...........................................................176
...no description..

H.1 Classes

H.1.1 Class Assumptions

H.1.1.1 Declaration

```java
public final class Assumptions
extends java.lang.Object
```

H.1.1.2 Methods

- **checkAssumptionsEqual**
  public void checkAssumptionsEqual( jive.PVC.Container.Assumptions b )

- **contains**
  public boolean contains( jive.PVC.Container.Triple t )
  - Usage
    * returns true, iff this Assumptions contain Triple t

- **elementAt**
  public Triple elementAt( int n )
  - Usage
* return the n-th assumption od this assumption set

- **elements**
  public Enumeration elements( )
  
  - **Usage**
  * returns the triples in this assumption set as enumeration

- **equals**
  public boolean equals( java.lang.Object o )
  
  - **Usage**
  * an equals Method for Assumptions

- **getSize**
  public int getSize( )
  
  - **Usage**
  * returns the size of the assumption set

- **isEmpty**
  public boolean isEmpty( )
  
  - **Usage**
  * returns true, iff Assumptionset is empty

- **read**
  public static Assumptions read( java.io.BufferedReader br )
  
  - **Usage**
  * reads Assumptions from the given BufferedReader.

- **toString**
  public String toString( )
  
  - **Usage**
  * returns the assumption set as a comma separated string of triples

- **write**
  public void write( java.io.PrintWriter sw )
  
  - **Usage**
  * writes these Assumptions into the PrintWriter

### H.1.2 Class CompRefString

#### H.1.2.1 Declaration

```java
public class CompRefString
    extends jive.PVC.Container.ContainerString
```
H.1.2.2 Constructors

- `CompRefString`
  
  ```java
  public CompRefString( jive.PC.Program.CompRef cr )
  ```

- `CompRefString`
  
  ```java
  public CompRefString( java.lang.String s )
  ```

H.1.2.3 Methods

- `getCompRef`
  
  ```java
  public CompRef getCompRef()```

H.1.3 Class `FormulaString`

H.1.3.1 Declaration

```java
public class FormulaString
extends jive.PVC.Container.ContainerString
```

H.1.3.2 Constructors

- `FormulaString`
  
  ```java
  public FormulaString( jive.PC.Container.Formula.Formula f )
  ```

- `FormulaString`
  
  ```java
  public FormulaString( java.lang.String s )
  ```

H.1.4 Class `LogicalVarString`

H.1.4.1 Declaration

```java
public class LogicalVarString
extends jive.PVC.Container.ContainerString
```

H.1.4.2 Constructors

- `LogicalVarString`
  
  ```java
  public LogicalVarString( java.lang.String s )
  ```

H.1.5 Class `Rule`
H.1.5.1 Declaration

```java
public abstract class Rule
extends java.lang.Object
```

H.1.5.2 Methods

- `call`
  ```java
  public ProofTreeNode call(java.lang.Object o)
  ```

- `call`
  ```java
  public ProofTreeNode call(java.lang.Object [] o)
  ```

- `call`
  ```java
  public ProofTreeNode call(java.lang.Object o1, java.lang.Object o2)
  ```

- `call`
  ```java
  public ProofTreeNode call(java.lang.Object o1, java.lang.Object o2, java.lang.Object o3)
  ```

- `call`
  ```java
  public ProofTreeNode call(java.lang.Object o1, java.lang.Object o2, java.lang.Object o3, java.lang.Object o4)
  ```

- `getBackwardParameters`
  ```java
  protected abstract Object getBackwardParameters()
  ```

  - **Usage**
    ```java
    * returns the parameters for the backward rule call without the prooftreenode
    ```

- `getContainer`
  ```java
  public ProofContainer getContainer()
  ```

- `getParameterCount`
  ```java
  public int getParameterCount()
  ```

- `getParameterDescription`
  ```java
  public String getParameterDescription(int i)
  ```

- `getParameterTypes`
  ```java
  public Class getParameterTypes()
  ```

- `getRuleName`
  ```java
  public String getRuleName()
  ```

- `getView`
  ```java
  public View getView()
  ```

- `invokeCallMethod`
  ```java
  public ProofTreeNode invokeCallMethod(java.lang.Object [] parameter)
  ```
H.1.6  *Class Tactic*

**H.1.6.1 Declaration**

```java
public abstract class Tactic
extends jive.PVC.Container.Rule
```

**H.1.6.2 Constructors**

- *Tactic*
  ```java
  public Tactic( jive.PVC.Container.ProofContainer c,
  jive.PVC.Container.View.View v )
  ```

**H.1.6.3 Methods**

- *getBackwardParameters*
  ```java
  protected final Object getBackwardParameters( )
  ```

H.1.7  *Class ProgVarString*

**H.1.7.1 Declaration**

```java
public class ProgVarString
extends jive.PVC.Container.ContainerString
```

**H.1.7.2 Constructors**

- *ProgVarString*
  ```java
  public ProgVarString( java.lang.String s )
  ```

H.1.8  *Class subst_backward*

**H.1.8.1 Declaration**

```java
public class subst_backward
extends jive.PVC.Container.Rule
```

**H.1.8.2 Constructors**

- *subst_backward*
  ```java
  public subst_backward( jive.PVC.Container.ProofContainer c,
  jive.PVC.Container.View.View v )
  ```
H.1.8.3 Methods

- `getBackwardParameters`
  protected Object getBackwardParameters() 

H.1.9 Class `call_backward`

H.1.9.1 Declaration

```java
public class call_backward
extends jive.PVC.Container.Rule
```

H.1.9.2 Constructors

- `call_backward`
  public call_backward(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.9.3 Methods

- `getBackwardParameters`
  protected Object getBackwardParameters() 

H.1.10 Class `ReqException`

This Exception is thrown, iff a requirement test fails. The text contains detailed information

H.1.10.1 Declaration

```java
public class ReqException
extends java.lang.Exception
```

H.1.10.2 Constructors

- `ReqException`
  public ReqException() 
  
- `ReqException`
  public ReqException(java.lang.String s )

H.1.10.3 Methods

- `toString`
  public String toString() 

H.1.11 Class all_backward

H.1.11.1 Declaration

```java
public class all_backward
extends jive.PVC.Container.Rule
```

H.1.11.2 Constructors

- `all_backward`
  ```java
  public all_backward( jive.PVC.Container.ProofContainer c,
  jive.PVC.Container.View.View v )
  ```

H.1.11.3 Methods

- `getBackwardParameters`
  ```java
  public Object getBackwardParameters() 
  ```

H.1.12 Class all_forward

H.1.12.1 Declaration

```java
public class all_forward
extends jive.PVC.Container.Rule
```

H.1.12.2 Constructors

- `all_forward`
  ```java
  public all_forward( jive.PVC.Container.ProofContainer c,
  jive.PVC.Container.View.View v )
  ```

H.1.12.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters() 
  ```

H.1.13 Class assumpt_elim_backward

H.1.13.1 Declaration

```java
public class assumpt_elim_backward
extends jive.PVC.Container.Rule
```
H.1.13.2 Constructors

• assumpt_elim_backward
  public assumpt_elim_backward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.13.3 Methods

• getBackwardParameters
  protected Object getBackwardParameters( )

H.1.14 Class assumpt_elim_forward

H.1.14.1 Declaration

public class assumpt_elim_forward
extends jive.PVC.Container.Rule

H.1.14.2 Constructors

• assumpt_elim_forward
  public assumpt_elim_forward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.14.3 Methods

• getBackwardParameters
  protected Object getBackwardParameters( )

H.1.15 Class ContextInfo

H.1.15.1 Declaration

public class ContextInfo
extends java.lang.Object

H.1.15.2 Methods

• getBackwardParameters
  public Object getBackwardParameters( )

• getRuleName
  public String getRuleName( )
### H.1.16 Class assumpt_introl_backward

#### H.1.16.1 Declaration

```java
public class assumpt_introl_backward
extends jive.PVC.Container.Rule
```

#### H.1.16.2 Constructors

- `assumpt_introl_backward`
  ```java
  public assumpt_introl_backward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

#### H.1.16.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

### H.1.17 Class assumpt_introl_forward

#### H.1.17.1 Declaration

```java
public class assumpt_introl_forward
extends jive.PVC.Container.Rule
```

#### H.1.17.2 Constructors

- `assumpt_introl_forward`
  ```java
  public assumpt_introl_forward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

#### H.1.17.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

### H.1.18 Class block_backward

#### H.1.18.1 Declaration

```java
public class block_backward
extends jive.PVC.Container.Rule
```
H.1.18.2 Constructors

- `block_backward`
  public `block_backward` (jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)

H.1.18.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters`( )

H.1.19 Class `block_forward`

H.1.19.1 Declaration

```java
public class block_forward
    extends jive.PVC.Container.Rule
```

H.1.19.2 Constructors

- `block_forward`
  public `block_forward` (jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)

H.1.19.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters`( )

H.1.20 Class `call_forward`

H.1.20.1 Declaration

```java
public class call_forward
    extends jive.PVC.Container.Rule
```

H.1.20.2 Constructors

- `call_forward`
  public `call_forward` (jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)
H.1.20.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`

H.1.21 Class `var_backward`

H.1.21.1 Declaration

```java
public class var_backward
extends jive.PVC.Container.Rule
```

H.1.21.2 Constructors

- `var_backward`
  public `var_backward`( jive.PVC.Container.ProofContainer `c`,
  jive.PVC.Container.View.View `v` )

H.1.21.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`

H.1.22 Class `Triple`

H.1.22.1 Declaration

```java
public class Triple
extends java.lang.Object
implements java.io.Serializable
```

H.1.22.2 Constructors

- `Triple`
  protected `Triple`( jive.PVC.Container.Formula.BooleanFormula `pre`,
  jive.PC.Program.CompRef `ref`, jive.PVC.Container.Formula.BooleanFormula `post` )
  - Usage
    * The constructors of `Triple`
H.1.22.3 Methods

- `checkTriplesEqual`
  ```java
  public void checkTriplesEqual( jive.PVC.Container.Triple t )
  ```
  - **Usage**
    * checks if two triples are structurally equal and throws an exception if not

- `equals`
  ```java
  public boolean equals( java.lang.Object o )
  ```
  - **Usage**
    * this method overrides Object.equals to correctly implement the comparison of two triples

- `getCompRef`
  ```java
  public CompRef getCompRef( )
  ```
  - **Usage**
    * return compref of triple

- `getPost`
  ```java
  public BooleanFormula getPost( )
  ```
  - **Usage**
    * return postcondition of triple

- `getPre`
  ```java
  public BooleanFormula getPre( )
  ```
  - **Usage**
    * return precondition of triple

- `read`
  ```java
  public static Triple read( java.io.BufferedReader br )
  ```
  - **Usage**
    * reads a Triple from the given BufferedReader.

- `toString`
  ```java
  public String toString( )
  ```
  - **Usage**
    * returns a String representation of this Triple

- `write`
  ```java
  public void write( java.io.PrintWriter sw )
  ```
  - **Usage**
    * writes this Triple into the PrintWriter
H.1.23  Class check_assign_axiom

H.1.23.1 Declaration

```java
public class check_assign_axiom
extends jive.PVC.Container.Rule
```

H.1.23.2 Constructors

- `check_assign_axiom`
  ```java
  public check_assign_axiom( jive.PVC.Container.ProofContainer c, 
jive.PVC.Container.View.View v )
  ```

H.1.23.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

H.1.24  Class check_assumpt_axiom

H.1.24.1 Declaration

```java
public class check_assumpt_axiom
extends jive.PVC.Container.Rule
```

H.1.24.2 Constructors

- `check_assumpt_axiom`
  ```java
  public check_assumpt_axiom( jive.PVC.Container.ProofContainer c, 
jive.PVC.Container.View.View v )
  ```

H.1.24.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

H.1.25  Class check_false_axiom

H.1.25.1 Declaration

```java
public class check_false_axiom
extends jive.PVC.Container.Rule
```
H.1.25.2 Constructors

- `check_false_axiom`
  public check_false_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.25.3 Methods

- `getBackwardParameters`
  protected Object getBackwardParameters( )

H.1.26 Class `check_field_read_axiom`

H.1.26.1 Declaration

```java
public class check_field_read_axiom
extends jive.PVC.Container.Rule
```

H.1.26.2 Constructors

- `check_field_read_axiom`
  public check_field_read_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.26.3 Methods

- `getBackwardParameters`
  protected Object getBackwardParameters( )

H.1.27 Class `check_field_write_axiom`

H.1.27.1 Declaration

```java
public class check_field_write_axiom
extends jive.PVC.Container.Rule
```

H.1.27.2 Constructors

- `check_field_write_axiom`
  public check_field_write_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
H.1.27.3 Methods

- **getBackwardParameters**
  
  protected Object getBackwardParameters()

H.1.28 Class *check_return_axiom*

H.1.28.1 Declaration

```java
public class check_return_axiom
extends jive.PVC.Container.Rule
```

H.1.28.2 Constructors

- **check_return_axiom**
  
  public check_return_axiom( jive.PVC.Container.ProofContainer c,
  jive.PVC.Container.View.View v )

H.1.28.3 Methods

- **getBackwardParameters**
  
  protected Object getBackwardParameters()

H.1.29 Class *class_backward*

H.1.29.1 Declaration

```java
public class class_backward
extends jive.PVC.Container.Rule
```

H.1.29.2 Constructors

- **class_backward**
  
  public class_backward( jive.PVC.Container.ProofContainer c,
  jive.PVC.Container.View.View v )

H.1.29.3 Methods

- **getBackwardParameters**
  
  protected Object getBackwardParameters()
H.1.30  *Class*  `class_forward`

### H.1.30.1 Declaration
```
public class class_forward
extends jive.PVC.Container.Rule
```

### H.1.30.2 Constructors

- `class_forward`  
  ```java
  public class_forward( jive.PVC.Container.ProofContainer  c,
                   jive.PVC.Container.View.View  v )
  ```

### H.1.30.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

H.1.31  *Class*  `conjunct_backward`

### H.1.31.1 Declaration
```
public class conjunct_backward
extends jive.PVC.Container.Rule
```

### H.1.31.2 Constructors

- `conjunct_backward`  
  ```java
  public conjunct_backward( jive.PVC.Container.ProofContainer  c,
                   jive.PVC.Container.View.View  v )
  ```

### H.1.31.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

H.1.32  *Class*  `conjunct_forward`

### H.1.32.1 Declaration
```
public class conjunct_forward
extends jive.PVC.Container.Rule
```
H.1.32.2  Constructors

- `conjunct_forward`
  public `conjunct_forward(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)`

H.1.32.3  Methods

- `getBackwardParameters`
  protected `Object getBackwardParameters()`

H.1.33  Class `disjunct_backward`

H.1.33.1  Declaration

```
public class disjunct_backward
extends jive.PVC.Container.Rule
```

H.1.33.2  Constructors

- `disjunct_backward`
  public `disjunct_backward(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)`

H.1.33.3  Methods

- `getBackwardParameters`
  protected `Object getBackwardParameters()`

H.1.34  Class `disjunct_forward`

H.1.34.1  Declaration

```
public class disjunct_forward
extends jive.PVC.Container.Rule
```

H.1.34.2  Constructors

- `disjunct_forward`
  public `disjunct_forward(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)`
H.1.34.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`

H.1.35 Class `ex_backward`

H.1.35.1 Declaration

```java
public class ex_backward
    extends jive.PVC.Container.Rule
```

H.1.35.2 Constructors

- `ex_backward`
  public `ex_backward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )`

H.1.35.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`

H.1.36 Class `ex_forward`

H.1.36.1 Declaration

```java
public class ex_forward
    extends jive.PVC.Container.Rule
```

H.1.36.2 Constructors

- `ex_forward`
  public `ex_forward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )`

H.1.36.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`
H.1.37 Class if_backward

H.1.37.1 Declaration
public class if_backward
extends jive.PVC.Container.Rule

H.1.37.2 Constructors

- if_backward
  public if_backward( jive.PVC.Container.ProofContainer c,
                    jive.PVC.Container.View.View v )

H.1.37.3 Methods

- getBackwardParameters
  protected Object getBackwardParameters( )

H.1.38 Class if_forward

H.1.38.1 Declaration
public class if_forward
extends jive.PVC.Container.Rule

H.1.38.2 Constructors

- if_forward
  public if_forward( jive.PVC.Container.ProofContainer c,
                    jive.PVC.Container.View.View v )

H.1.38.3 Methods

- getBackwardParameters
  protected Object getBackwardParameters( )

H.1.39 Class implementation_backward

H.1.39.1 Declaration
public class implementation_backward
extends jive.PVC.Container.Rule
### H.1.39.2 Constructors

- `implementation_backward`
  
  ```java
  public implementation_backward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

### H.1.39.3 Methods

- `getBackwardParameters`
  
  ```java
  protected Object getBackwardParameters( )
  ```

### H.1.40 Class `implementation_forward`

#### H.1.40.1 Declaration

```java
public class implementation_forward
extends jive.PVC.Container.Rule
```

#### H.1.40.2 Constructors

- `implementation_forward`
  
  ```java
  public implementation_forward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

#### H.1.40.3 Methods

- `getBackwardParameters`
  
  ```java
  protected Object getBackwardParameters( )
  ```

### H.1.41 Class `inst_assign_axiom`

#### H.1.41.1 Declaration

```java
public class inst_assign_axiom
extends jive.PVC.Container.Rule
```

#### H.1.41.2 Constructors

- `inst_assign_axiom`
  
  ```java
  public inst_assign_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```
H.1.41.3 Methods

- getBackwardParameters
  protected Object getBackwardParameters()

H.1.42 Class inst_assumpt_axiom

H.1.42.1 Declaration

```java
public class inst_assumpt_axiom
extends jive.PVC.Container.Rule
```

H.1.42.2 Constructors

- inst_assumpt_axiom
  public inst_assumpt_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.42.3 Methods

- getBackwardParameters
  protected Object getBackwardParameters()

H.1.43 Class inst_false_axiom

H.1.43.1 Declaration

```java
public class inst_false_axiom
extends jive.PVC.Container.Rule
```

H.1.43.2 Constructors

- inst_false_axiom
  public inst_false_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.43.3 Methods

- getBackwardParameters
  protected Object getBackwardParameters()
H.1.44  Class inst_field_read_axiom

H.1.44.1 Declaration

public class inst_field_read_axiom
extends jive.PVC.Container.Rule

H.1.44.2 Constructors

- inst_field_read_axiom
  public inst_field_read_axiom ( jive.PVC.Container.ProofContainer c,
                              jive.PVC.Container.View.View v )

H.1.44.3 Methods

- getBackwardParameters
  protected Object getBackwardParameters ( )

H.1.45  Class inst_field_write_axiom

H.1.45.1 Declaration

public class inst_field_write_axiom
extends jive.PVC.Container.Rule

H.1.45.2 Constructors

- inst_field_write_axiom
  public inst_field_write_axiom ( jive.PVC.Container.ProofContainer c,
                              jive.PVC.Container.View.View v )

H.1.45.3 Methods

- getBackwardParameters
  protected Object getBackwardParameters ( )

H.1.46  Class inst_return_axiom

H.1.46.1 Declaration

public class inst_return_axiom
extends jive.PVC.Container.Rule
H.1.46.2 Constructors

- inst\_return\_axiom
  
  ```java
  public inst\_return\_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

H.1.46.3 Methods

- getBackward\_Parameters
  
  ```java
  protected Object getBackward\_Parameters( )
  ```

H.1.47 Class inv\_backward

H.1.47.1 Declaration

```java
public class inv\_backward
extends jive.PVC.Container.Rule
```

H.1.47.2 Constructors

- inv\_backward
  
  ```java
  public inv\_backward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

H.1.47.3 Methods

- getBackward\_Parameters
  
  ```java
  protected Object getBackward\_Parameters( )
  ```

H.1.48 Class inv\_forward

H.1.48.1 Declaration

```java
public class inv\_forward
extends jive.PVC.Container.Rule
```

H.1.48.2 Constructors

- inv\_forward
  
  ```java
  public inv\_forward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```
H.1.48.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`

H.1.49 Class `var_forward`

H.1.49.1 Declaration
```
public class var_forward
extends jive.PVC.Container.Rule
```

H.1.49.2 Constructors

- `var_forward`
  public `var_forward` (jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)

H.1.49.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`

H.1.50 Class `invocation_backward`

H.1.50.1 Declaration
```
public class invocation_backward
extends jive.PVC.Container.Rule
```

H.1.50.2 Constructors

- `invocation_backward`
  public `invocation_backward` (jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)

H.1.50.3 Methods

- `getBackwardParameters`
  protected Object `getBackwardParameters()`
H.1.51  

Class  

invocation\_forward

H.1.51.1  

Declaration

public class invocation\_forward
extends jive.PVC.Container.Rule

H.1.51.2  

Constructors

- invocation\_forward
  
  public invocation\_forward(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)

H.1.51.3  

Methods

- getBackwardParameters
  
  protected Object getBackwardParameters()

H.1.52  

Class  

seq\_back\_backward

H.1.52.1  

Declaration

public class seq\_back\_backward
extends jive.PVC.Container.Rule

H.1.52.2  

Constructors

- seq\_back\_backward
  
  public seq\_back\_backward(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)

H.1.52.3  

Methods

- getBackwardParameters
  
  protected Object getBackwardParameters()

H.1.53  

Class  

seq\_front\_backward

H.1.53.1  

Declaration

public class seq\_front\_backward
extends jive.PVC.Container.Rule
H.1.53.2 Constructors

- `seq_front_backward`
  public `seq_front_backward` ( `jive.PVC.Container.ProofContainer c`, `jive.PVC.Container.View.View v` )

H.1.53.3 Methods

- `getBackwardParameters`
  protected `Object getBackwardParameters()`

H.1.54 Class `seq_forward`

H.1.54.1 Declaration

public class `seq_forward`
extends `jive.PVC.Container.Rule`

H.1.54.2 Constructors

- `seq_forward`
  public `seq_forward` ( `jive.PVC.Container.ProofContainer c`, `jive.PVC.Container.View.View v` )

H.1.54.3 Methods

- `getBackwardParameters`
  protected `Object getBackwardParameters()`

H.1.55 Class `static_invocation_backward`

H.1.55.1 Declaration

public class `static_invocation_backward`
extends `jive.PVC.Container.Rule`

H.1.55.2 Constructors

- `static_invocation_backward`
  public `static_invocation_backward` ( `jive.PVC.Container.ProofContainer c`, `jive.PVC.Container.View.View v` )
H.1.55.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters()
  ```

H.1.56 Class `static_invocation_forward`

H.1.56.1 Declaration
```java
public class static_invocation_forward
extends jive.PVC.Container.Rule
```

H.1.56.2 Constructors

- `static_invocation_forward`
  ```java
  public static_invocation_forward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

H.1.56.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters()
  ```

H.1.57 Class `strength_backward`

H.1.57.1 Declaration
```java
public class strength_backward
extends jive.PVC.Container.Rule
```

H.1.57.2 Constructors

- `strength_backward`
  ```java
  public strength_backward( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

H.1.57.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters()
  ```
H.1.58  

**Class check_cast_axiom**

### Declaration

```java
public class check_cast_axiom
extends jive.PVC.Container.Rule
```

### Constructors

- `check_cast_axiom`
  ```java
  public check_cast_axiom(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)
  ```

### Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters()
  ```

H.1.59  

**Class check_empty_axiom**

### Declaration

```java
public class check_empty_axiom
extends jive.PVC.Container.Rule
```

### Constructors

- `check_empty_axiom`
  ```java
  public check_empty_axiom(jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v)
  ```

### Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters()
  ```

H.1.60  

**Class subst_forward**

### Declaration

```java
public class subst_forward
extends jive.PVC.Container.Rule
```
H.1.60.2 Constructors

- `subst_forward`
  public `subst_forward` ( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.60.3 Methods

- `getBackwardParameters`
  protected `Object getBackwardParameters()`

H.1.61 Class `strength_forward`

H.1.61.1 Declaration

```java
public class strength_forward
extends jive.PVC.Container.Rule
```

H.1.61.2 Constructors

- `strength_forward`
  public `strength_forward` ( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )

H.1.61.3 Methods

- `getBackwardParameters`
  protected `Object getBackwardParameters()`

H.1.62 Class `subtype_backward`

H.1.62.1 Declaration

```java
public final class subtype_backward
extends jive.PVC.Container.Rule
```

H.1.62.2 Constructors

- `subtype_backward`
  public `subtype_backward` ( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
H.1.62.3 Methods

- `getBackwardParameters`
  
  ```java
  protected Object getBackwardParameters()
  ```

- `getParameterDescription`
  
  ```java
  public String getParameterDescription( int i )
  ```

H.1.63 Class `subtype_forward`

H.1.63.1 Declaration

```java
public class subtype_forward

extends jive.PVC.Container.Rule
```

H.1.63.2 Constructors

- `subtype_forward`
  
  ```java
  public subtype_forward( jive.PVC.Container.ProofContainer c,
    jive.PVC.Container.View.View v )
  ```

H.1.63.3 Methods

- `getBackwardParameters`
  
  ```java
  protected Object getBackwardParameters()
  ```

H.1.64 Class `weak_backward`

H.1.64.1 Declaration

```java
public class weak_backward

extends jive.PVC.Container.Rule
```

H.1.64.2 Constructors

- `weak_backward`
  
  ```java
  public weak_backward( jive.PVC.Container.ProofContainer c,
    jive.PVC.Container.View.View v )
  ```

H.1.64.3 Methods

- `getBackwardParameters`
  
  ```java
  protected Object getBackwardParameters()
  ```
H.1.65  Class weak_forward

H.1.65.1 Declaration

```java
public class weak_forward
extends jive.PVC.Container.Rule
```

H.1.65.2 Constructors

- `weak_forward`
  ```java
  public weak_forward( jive.PVC.Container.ProofContainer c, 
  jive.PVC.Container.View.View v )
  ```

H.1.65.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

H.1.66  Class while_backward

H.1.66.1 Declaration

```java
public class while_backward
extends jive.PVC.Container.Rule
```

H.1.66.2 Constructors

- `while_backward`
  ```java
  public while_backward( jive.PVC.Container.ProofContainer c, 
  jive.PVC.Container.View.View v )
  ```

H.1.66.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters( )
  ```

H.1.67  Class while_forward

H.1.67.1 Declaration

```java
public class while_forward
extends jive.PVC.Container.Rule
```
H.1.67.2 Constructors

- `while_forward`
  ```java
  public while_forward( jive.PVC.Container.ProofContainer c, 
                      jive.PVC.Container.View.View v )
  ```

H.1.67.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters()
  ```

H.1.68 Class Sequent

H.1.68.1 Declaration

```java
public class Sequent 
extends java.lang.Object 
implements java.io.Serializable
```

H.1.68.2 Constructors

- `Sequent`
  ```java
  protected Sequent( jive.PVC.Container.Assumptions a, 
                     jive.PVC.Container.Formula.BooleanFormula pre, 
                     jive.PC.Program.CompRef ref, 
                     jive.PVC.Container.Formula.BooleanFormula post )
  ```
  - Usage
    * builds a sequent with the given date

- `Sequent`
  ```java
  protected Sequent( jive.PVC.Container.Assumptions a, 
                     jive.PVC.Container.Triple t )
  ```
  - Usage
    * builds a sequent with the given assumptions and the given triple

- `Sequent`
  ```java
  protected Sequent( jive.PVC.Container.Formula.BooleanFormula pre, 
                     jive.PC.Program.CompRef ref, 
                     jive.PVC.Container.Formula.BooleanFormula post )
  ```
  - Usage
    * builds a sequent with the given date
• **Sequent**
  
  protected *Sequent*(*jive.PVC.Container.Triple t*)
  
  – **Usage**
  
  * builds a sequent with the given triple and empty assumptions

### H.16.83 Methods

• **checkSequencesEqual**

  public *void* *checkSequencesEqual*(*jive.PVC.Container.Sequent s*)

  – **Usage**

  * checks if two sequents are structurally equal and throws an exception if not

• **equals**

  public *boolean* *equals*(*java.lang.Object o*)

  – **Usage**

  * return true iff this and sqn are structurally equal

• **getAssumptions**

  public *Assumptions* *getAssumptions*(* )

  – **Usage**

  * returns the assumptions of the sequent attribute

• **getCompRef**

  public *CompRef* *getCompRef*(* )

  – **Usage**

  * returns the CompRef of the attribute triple

• **getPost**

  public *BooleanFormula* *getPost*(* )

  – **Usage**

  * returns the postcondition of the sequent attribute

• **getPre**

  public *BooleanFormula* *getPre*(* )

  – **Usage**

  * returns the precondition of the sequent attribute

• **getTriple**

  protected *Triple* *getTriple*(* )

  – **Usage**

  * returns the Triple of attribute sequent
• **read**
  public static Sequent read( java.io.BufferedReader br )
  
  – **Usage**
  * reads Assumptions from the given BufferedReader.

• **removeAssumption**
  protected Sequent removeAssumption( jive.PVC.Container.Triple t )
  
  – **Usage**
  * removes an assumption from this assumptions, no sideeffect

• **toString**
  public String toString( )
  
  – **Usage**
  * return a String representation of this sequent

• **write**
  public void write( java.io.PrintWriter sw )
  
  – **Usage**
  * writes these Assumptions into the PrintWriter

### H.1.69 Class inst_cast_axiom

#### H.1.69.1 Declaration

```java
public class inst_cast_axiom
    extends jive.PVC.Container.Rule
```

#### H.1.69.2 Constructors

• **inst_cast_axiom**
  ```java
  public inst_cast_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

#### H.1.69.3 Methods

• **getBackwardParameters**
  ```java
  protected Object getBackwardParameters( )
  ```

### H.1.70 Class inst_empty_axiom
H.1.70.1 Declaration

```java
public class inst_empty_axiom extends jive.PVC.Container.Rule
```

H.1.70.2 Constructors

- `inst_empty_axiom`
  ```java
  public inst_empty_axiom( jive.PVC.Container.ProofContainer c, jive.PVC.Container.View.View v )
  ```

H.1.70.3 Methods

- `getBackwardParameters`
  ```java
  protected Object getBackwardParameters() ( )
  ```

H.1.71 Class ProofTreeNode

H.1.71.1 Declaration

```java
public class ProofTreeNode extends java.lang.Object implements javax.swing.tree.TreeNode
```

H.1.71.2 Fields

- public int refereby

H.1.71.3 Methods

- `checkOpenProofSlot`
  ```java
  public void checkOpenProofSlot( )
  ```
  ```java
  Usage
  * helper method, checks if a ProofTreeNode is an open proofslot
  ```

- `checkProofTreeRoot`
  ```java
  public void checkProofTreeRoot( )
  ```
  ```java
  Usage
  * helper method, checks if a ProofTree is correctly registered within this container
  ```
- **children**
  
  ```java
  public Enumeration children()
  ```

- **concat**

  ```java
  public ProofTreeNode concat( jive.PVC.Container.ProofTreeNode lower )
  ```
  
  - **Usage**
    
    * this operation merges a prooftree into an open proof slot. the root node of the prooftree and the open proof slot must be 'equal'. if one of them is a goal, the remaining node is a goal.

- **containsGoalOrHoareLemma**

  ```java
  public boolean containsGoalOrHoareLemma()
  ```
  
  - **Usage**
    
    * checks if the subtree spanned by this contains atleast one goal

- **getAllowsChildren**

  ```java
  public boolean getAllowsChildren()
  ```

- **getAssumptions**

  ```java
  public Assumptions getAssumptions()
  ```
  
  - **Usage**
    
    * returns the assumptions of the sequent attribute

- **getChildAt**

  ```java
  public TreeNode getChildAt( int childIndex )
  ```

- **getChildCount**

  ```java
  public int getChildCount()
  ```

- **getChildren**

  ```java
  public ProofTreeNode getChildren()
  ```
  
  - **Usage**
    
    * returns the children of this ProofTreeNode-Object

- **getCompRef**

  ```java
  public CompRef getCompRef()
  ```
  
  - **Usage**
    
    * returns the CompRef of the attribute sequent

- **getContextInfo**

  ```java
  public ContextInfo getContextInfo()
  ```
  
  - **Usage**
    
    * returns the ContextInfo of this Node
• getHoareLemma
  public ProofTreeNode getHoareLemma( )

  – Usage
  * returns the referred lemma node

• getId
  public String getId( )

  – Usage
  * return the id of this ProofTreeNode

• getIndex
  public int getIndex( javax.swing.tree.TreeNode node )

• getLemma
  public BooleanFormula getLemma( )

  – Usage
  * return the lemma of this ProofTreeNode

• getLemmaState
  public String getLemmaState( )

  – Usage
  * returns the state of the lemma

• getParent
  public TreeNode getParent( )

  – Usage
  * the methods of the ProofTree interface

• getPost
  public BooleanFormula getPost( )

  – Usage
  * returns the postcondition of the sequent attribute

• getPre
  public BooleanFormula getPre( )

  – Usage
  * returns the precondition of the sequent attribute

• getProofTreeNodes
  public Vector getProofTreeNodes( )

  – Usage
  * returns all prooftreenodes with this prooftreenode as root as result
• `getReferredHoareLemmataInSubtree`
  public Enumeration getReferredHoareLemmataInSubtree()  
    
    - Usage
    * returns all HoareLemmata, which are referred within the subtree spanned by this

• `getRoot`
  public ProofTreeNode getRoot()  
    
    - Usage
    * returns the root of the ProofTree containing this ProofTreeNode

• `getSequent`
  public Sequent getSequent()  
    
    - Usage
    * returns the sequent attribute

• `hasChildren`
  public boolean hasChildren()  

• `hasLemma`
  public boolean hasLemma()  
    
    - Usage
    * returns true, iff this proofTreeNode has a lemma

• `isGoal`
  public boolean isGoal()  
    
    - Usage
    * the set/get pair for the goal-flag

• `isHoareLemma`
  public boolean isHoareLemma()  
    
    - Usage
    * returns true, iff this node is a Hoare Lemma Node

• `isLeaf`
  public boolean isLeaf()  

• `isOpenProofSlot`
  public boolean isOpenProofSlot()  
    
    - Usage
    * checks if this proofTreeNode is an open proof slot, i.e. it has no children, the sequent.closed-flag is not set, and has no lemma
• isRoot
   public boolean isRoot( )

• isSequentClosed
   public boolean isSequentClosed() {
      – Usage
      * returns true, iff all sequents with this prooftree node as root are closed

• makeHoareLemma
   public ProofTreeNode makeHoareLemma() {
      – Usage
      * lets this node be a Hoare-Lemma Node

• reaches
   public boolean reaches( jive.PVC.Container.ProofTreeNode ptn ) {
      – Usage
      * returns true, iff prooftree node ptn is reachable in the prooftree spanned by this node via prooftree edges or the lemma edge

• read
   public static ProofTreeNode read( javax.swing.tree.TreeNode parent, java.io.BufferedReader br ) {
      – Usage
      * reads ProofTreeNode from the given BufferedReader.

• refersHoareLemma
   public boolean refersHoareLemma() {
      – Usage
      * returns true, if the proof of this node refers to a lemma

• referToHoareLemma
   public void referToHoareLemma() {
      – Usage
      * lets this node refer to a lemma.

• removeRoot
   public void removeRoot() {
      – Usage
      * removes the root of a prooftree of possible

• rmHoareLemma
   public ProofTreeNode rmHoareLemma() {

split

public ProofTreeNode split( )

- Usage
  * splits a prooftree at a given prooftreenode ptn.

toString

public String toString( )

write

public void write( java.io.PrintWriter sw )

- Usage
  * writes this ProofTreeNode into the PrintWriter

H.1.72  Class ProofContainer

H.1.72.1  Declaration

public final class ProofContainer
extends java.lang.Object
implements javax.swing.tree.TreeNode

H.1.72.2  Fields

- private static Class tactics
- private static Class axioms
- private static Class foperations
- private static Class boperations

H.1.72.3  Constructors

- ProofContainer
  public ProofContainer( java.lang.String path, java.lang.String basename )

- Usage
  * the one and only constructor of ProofContainer
H.1.72.4 Methods

- **addView**
  public void addView( jive.PVC.Container.View.View v )
  
  - **Usage**
    * registers a new view for this container

- **bottom**
  public ProofTreeNode bottom( jive.PVC.Container.ProofTreeNode ptn )

- **changeTreeSeq**
  public void changeTreeSeq( int i, int j )
  
  - **Usage**
    * changes the sequence of two prooftrees, throws an exception iff
      0>=i,j>=prooftrees.size

- **checkLogicalVar**
  public void checkLogicalVar( jive.PVC.Container.LogicalVarString name )
  
  - **Usage**
    * checks, iff name exists in symboltable if not it throws a ReqException

- **checkLogicalVar**
  public void checkLogicalVar( java.lang.String name )

- **children**
  public Enumeration children( )
  
  - **Usage**
    * Returns the children of the reciever as an Enumeration.

- **closeProject**
  public void closeProject( )
  
  - **Usage**
    * close all views and shut down this project

- **dispatchMessage**
  public void dispatchMessage( java.lang.String text )
  
  - **Usage**
    * dispatches a message to all registered views

- **down**
  public ProofTreeNode down( jive.PVC.Container.ProofTreeNode ptn )
• **generateLemmaTheory**
  public void generateLemmaTheory( )

• **getAllowsChildren**
  public boolean getAllowsChildren( )
  
  – **Usage**
  * Returns true if the receiver allows children.

• **getBaseName**
  public String getBaseName( )
  
  – **Usage**
  * returns the basename of this project

• **getChildAt**
  public TreeNode getChildAt( int childIndex )
  
  – **Usage**
  * Returns the child TreeNode at index childIndex.

• **getChildCount**
  public int getChildCount( )
  
  – **Usage**
  * returns the number of childs of this ProofTreeNode, from TreeNode interface

• **getIndex**
  public int getIndex( javax.swing.tree.TreeNode node )
  
  – **Usage**
  * Returns the index of node in the receivers children.

• **getLemmas**
  public String getLemmas( )
  
  – **Usage**
  * returns all lemmas of all proof trees currently within the proof container

• **getNativeAt**
  public Sequent getNativeAt( int i )

• **getNativeCount**
  public int getNativeCount( )

• **getOpenProofSlots**
  public Vector getOpenProofSlots( )
  
  – **Usage**
  * returns all open proof slots of this container
• **getParent**
  
  public TreeNode getParent()

  - **Usage**
    
    * returns the parent of this Node, from TreeNode interface

• **getPathName**
  
  public String getPathName()

  - **Usage**
    
    * returns the path name of this project

• **getProofTreeNodes**
  
  public Vector getProofTreeNodes()

• **getProofTrees**
  
  public Vector getProofTrees()

• **getSymboltable**
  
  public Hashtable getSymboltable()

  - **Usage**
    
    * return the symboltable of logical variables

• **getViews**
  
  public Vector getViews()

• **insertAxiom**
  
  public ProofTreeNode insertAxiom(

  - **Usage**
    
    * insert a proved sequent to the container

• **insertAxiom**
  
  public ProofTreeNode insertAxiom( jive.PVC.Container.Sequent s)

  - **Usage**
    
    * insert a proven sequent to the container

• **insertGoal**
  
  public ProofTreeNode insertGoal(

  - **Usage**
* insert a sequent to the container which remains to prove

- **insertGoal**
  public ProofTreeNode insertGoal( jive.PVC.Container.Sequent s )
  
  - **Usage**
    * insert a sequent to the container which remains to prove

- **insertLogicalVar**
  public void insertLogicalVar( java.lang.String name, java.lang.String sort )
  
  - **Usage**
    * inserts a logical variable into the symboltable

- **insertNative**
  
  - **Usage**
    * insert a native to the container

- **insertNative**
  public void insertNative( jive.PVC.Container.Sequent s )
  
  - **Usage**
    * insert a native to the container

- **insertProofTree**
  
  - **Usage**
    * insert a prooftree wit a given sequent to the container

- **insertProofTree**
  public ProofTreeNode insertProofTree( jive.PVC.Container.Sequent s )

- **isLeaf**
  public boolean isLeaf( )
  
  - **Usage**
    * Returns true if the receiver is a leaf.

- **loadProofTrees**
  public void loadProofTrees( java.io.BufferedReader in )
• `openDefaultView`
  
  public void `openDefaultView()`
  
  - **Usage**
    - * opens a default view to this ProofContainer, at the moment this is a TreeView

• `parseBooleanFormula`
  
  public `BooleanFormula` `parseBooleanFormula`(  
  `jive.PVC.Container.ContainerString` f, `jive.PC.Program.CompRef` cr )

• `parseBooleanFormula`
  
  public `BooleanFormula` `parseBooleanFormula`(  
  `java.lang.String` f,  
  `jive.PC.Program.CompRef` cr )
  
  - **Usage**
    - * helper method, parses a string to a BooleanFormula, throws Exception if somethings wrong

• `raiseReqException`
  
  public static void `raiseReqException`(  
  `java.lang.String` s )
  
  - **Usage**
    - * throws a new ReqException with a given message s

• `removeSubtree`
  
  public void `removeSubtree`(  
  `jive.PVC.Container.ProofTreeNode` ptn )
  
  - **Usage**
    - * removes all subtrees of a node

• `removeTree`
  
  public void `removeTree`(  
  `jive.PVC.Container.ProofTreeNode` t )
  
  - **Usage**
    - * removes a complete prooftere from this proofcontainer and from the nametable

• `removeView`
  
  public void `removeView`(  
  `jive.PVC.Container.View.View` v )
  
  - **Usage**
    - * removes the view v from this container, i.e. it is not notified if the prooftere changes returns true, iff the view v was really removed

• `saveProofTrees`
  
  public void `saveProofTrees`(  
  `java.io.PrintWriter` sw )
• setLemmaStatus
  public void setLemmaStatus( java.lang.String lemma, java.lang.String state )

• setNextPosition
  public void setNextPosition( jive.PVC.Container.ProofTreeNode p )
  - Usage
    * set the node after whose root the next prooftree node should be inserted

• substitute
  public BooleanFormula substitute( jive.PVC.Container.Formula.BooleanFormula f,
  jive.PVC.Container.ContainerString a,
  jive.PVC.Container.ContainerString b, jive.PC.Program.CompRef cr )

• substitute
  public BooleanFormula substitute( jive.PVC.Container.Formula.BooleanFormula f,
  jive.PVC.Container.ContainerString a,
  jive.PVC.Container.ContainerString b, jive.PC.Program.CompRef from,
  jive.PC.Program.CompRef to )

• substitute
  public BooleanFormula substitute( jive.PVC.Container.Formula.BooleanFormula f,
  jive.PVC.Container.ContainerString a, java.lang.String b,
  jive.PC.Program.CompRef cr )

• substitute
  public BooleanFormula substitute( jive.PVC.Container.Formula.BooleanFormula f,
  jive.PVC.Container.ContainerString a, java.lang.String b,
  jive.PC.Program.CompRef from, jive.PC.Program.CompRef to )

• substitute
  public BooleanFormula substitute( jive.PVC.Container.Formula.BooleanFormula f, java.lang.String a,
  jive.PVC.Container.ContainerString b, jive.PC.Program.CompRef cr )

• substitute
  public BooleanFormula substitute( jive.PVC.Container.Formula.BooleanFormula f, java.lang.String a,
  jive.PVC.Container.ContainerString b, jive.PC.Program.CompRef from,
  jive.PC.Program.CompRef to )
  - Usage
    * helper method, performs a substitution of a for b in f
• substitute
  public BooleanFormula substitute( 
    jive.PVC.Container.Formula.BooleanFormula f, java.lang.String a, 
    java.lang.String b, jive.PC.Program.CompRef cr )
  
• substitute
  public BooleanFormula substitute( 
    jive.PVC.Container.Formula.BooleanFormula f, java.lang.String a, 
    java.lang.String b, jive.PC.Program.CompRef from, 
    jive.PC.Program.CompRef to )
  
• top
  public ProofTreeNode top( jive.PVC.Container.ProofTreeNode ptn )
  
• up
  public ProofTreeNode up( jive.PVC.Container.ProofTreeNode ptn )