API Documentation

Table of Contents

Frontend

Parser API

Overview: The Parser API describes methods available to the parser of the Simple Program Analyser (SPA). The main method, parseSimple, is the main entry point to the program, where a SIMPLE program string is input for analysis.

- [parseSimple]
- [parseArithmeticExpressions]

VOID parseSimple(**STRING** rawProgram) throws **SYNTAX_ERROR**;

Description: Takes in a SIMPLE program string so the PKB can be populated with entries. If the program string is not in valid SIMPLE syntax, or contains semantic errors, an ERROR will be thrown.
Normal behaviour: The program is parsed successfully, and entries stored in the PKB for queries.
Abnormal behaviour: If there is a syntax error in the SIMPLE program, a SYNTAX_ERROR will be thrown. An empty string results in a SYNTAX_ERROR as well, as SIMPLE programs require one or more procedures. Once Parser determines that the program is syntactically incorrect, parsing of the incorrect program is stopped indefinitely.

```
ARITHMETIC_EXPRESSION parseArithmeticExpression(STRING_LIST lexedArithmeticExpression) throws SYNTAX_ERROR;
```

Description: Takes in a lexed SIMPLE binary expression, and returns the root node of the Abstract Syntax Tree (AST) that represents the binary expression. This method may be used to pattern-match queries.

Normal behaviour: The lexed binary expression is parsed successfully, and the AST representing the binary expression is returned.
Abnormal behaviour: If there is a syntax error in the SIMPLE binary expression, a **SYNTAX_ERROR** will be thrown. An empty list will return **SYNTAX_ERROR** as well. Once Parser determines that the expression is syntactically incorrect, parsing of the incorrect program is stopped indefinitely.

Design Extractor API

Overview: The Design Extractor API describes methods available to the Design Extractor of the Simple Program Analyser (SPA). In the Design Extractor, program design entity relationships are identified and stored in the PKB. The main method, extractDesign, provides the inputs required by the Design Extractor to determine program design entity relationships, namely an Abstract Syntax Tree (AST) of a SIMPLE program.

- [extractDesign]

```
VOID extractDesign(PROGRAM_NODE rootNode) throws SEMANTIC_ERROR;
```

Description: Takes in a SIMPLE AST and walks the tree, identifying the presence of important relationships between program design entities. If the program contains semantic errors, a SEMANTIC_ERROR will be thrown.
Normal behaviour: The AST represents a semantically valid SIMPLE program, and the Design Extractor stores program design entity relationships in the PKB for queries.
Abnormal behaviour: If there is a semantic error in the SIMPLE program represented by the AST, a SEMANTIC_ERROR will be thrown. The design extractor will immediately cease operations, discarding the rest of the program that has not been analysed yet.

PKB

Abstract Syntax Tree (AST) API

Overview: The AST API describes the methods available to construct an Abstract Syntax Tree in the Simple Program Analyser (SPA).

- [createAssignNode]
- [createCallNode]
- [createIfNode]
- [createPrintNode]
- [createProcedureNode]
- [createProgramNode]
- [createReadNode]
- [createStmtlstNode]

- [createWhileNode]

`ASSIGNMENT_STATEMENT_NODE createAssignNode(STATEMENT_NUMBER sn, VARIABLE var, EXPRESSION expr);`

Description: Creates and returns an **ASSIGNMENT_STATEMENT_NODE** with `var` and `expr` as the children, and `sn` as its statement number.

`CALL_STATEMENT_NODE createCallNode(STATEMENT_NUMBER sn, NAME procName);`

Description: Creates and returns a **CALL_STATEMENT_NODE** with `procName` as the child, and `sn` as its statement number.

`IF_STATEMENT_NODE createIfNode(STATEMENT_NUMBER sn, CONDITIONAL_EXPRESSION predicate, STMTLST_NODE leftStatementList, STMTLST_NODE rightStatementList);`

Description: Creates and returns an **IF_STATEMENT_NODE** with the condition `predicate`, `leftStatementList` and `rightStatementList` as the children, and `sn` as its statement number.

`PRINT_STATEMENT_NODE createPrintNode(STATEMENT_NUMBER sn, VARIABLE var);`

Description: Creates and returns a **PRINT_STATEMENT_NODE** with `var` as the child, and `sn` as its statement number.

`PROCEDURE_NODE createProcedureNode(NAME procedureName, STMTLST_NODE stmtlstNode);`

Description: Creates and returns a **PROCEDURE_NODE** with `stmtlstNode` as the child, and `procedureName` as the name of the procedure.

`PROGRAM_NODE createProgramNode(NAME programName, PROCEDURE_NODE_LIST procedureNodes);`

Description: Creates and returns a **PROGRAM_NODE** with `procedureNodes` as the child in a **PROCEDURE_NODE_LIST** form, and `programName` as the name of the program.

`READ_STATEMENT_NODE createReadNode(STATEMENT_NUMBER sn, VARIABLE var);`

Description: Creates and returns a **READ_STATEMENT_NODE** with `var` as the child, and `sn` as its statement number.

`STMTLST_NODE createStmtlstNode(STATEMENT_NODE_LIST statementNodes);`

Description: Creates and returns a **STMTLST_NODE** with `statementNodes` as its children;

```
WHILE_STATEMENT_NODE createWhileNode(STATEMENT_NUMBER sn, CONDITIONAL_EXPRESSION predicate,
STMTLST_NODE statementList);
```

Description: Creates and returns an **WHILE_STATEMENT_NODE** with the condition `predicate`, `statementList` as its children, and `sn` as its statement number.

Expressions API

Overview: The Expressions API describes the methods available to create Expression representations in the Simple Program Analyser (SPA).

- [createAndExpr]
- [createDivExpr]
- [createEqExpr]
- [createGtExpr]
- [createGteExpr]
- [createLtExpr]
- [createLteExpr]
- [createMinusExpr]
- [createModExpr]
- [createNotExpr]
- [createOrExpr]
- [createPlusExpr]
- [createRefExpr]
- [createTimesExpr]

```
AND_EXPRESSION createAndExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);
```

Description: Creates and returns an **AND_EXPRESSION** where the truthy value depends on both `leftExpr` and the `rightExpr`.

```
ARITHMETIC_EXPRESSION createDivExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);
```

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is divided by the `rightExpr`.

```
RELATIONAL_EXPRESSION createGtExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);
```

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is equal to the `rightRelFactor`.

`RELATIONAL_EXPRESSION createGtExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);`

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is greater than the `rightRelFactor`.

`RELATIONAL_EXPRESSION createGteExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);`

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is greater than or equals to the `rightRelFactor`.

`RELATIONAL_EXPRESSION createLtExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);`

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is lesser than the `rightRelFactor`.

`RELATIONAL_EXPRESSION` **createLteExpr(EXPRESSION leftRelFactor, EXPRESSION rightRelFactor);**

Description: Creates and returns a **RELATIONAL_EXPRESSION** where the `leftRelFactor` is lesser than or equals to the `rightRelFactor`.

`ARITHMETIC_EXPRESSION createMinusExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);`

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is divided by the `rightExpr`.

`ARITHMETIC_EXPRESSION createModExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);`

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is mod by the `rightExpr`.

`NOT_EXPRESSION createNotExpr(CONDITIONAL_EXPRESSION expr);`

Description: Creates and returns an **NOT_EXPRESSION** with the negated value of `expr`.

`OR_EXPRESSION createOrExpr(CONDITIONAL_EXPRESSION leftExpr, CONDITIONAL_EXPRESSION rightExpr);`

Description: Creates and returns an **OR_EXPRESSION** where the truthy value depends on either `leftExpr` or the `rightExpr`.

`ARITHMETIC_EXPRESSION createPlusExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);`

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is added to the `rightExpr`.

```
REFERENCE_EXPRESSION createRefExpr(BASIC_DATA_TYPE basicData);
```

Description: Creates and returns a **REFERENCE_EXPRESSION** based on `basicData`.

```
ARITHMETIC_EXPRESSION createTimesExpr(EXPRESSION leftExpr, EXPRESSION rightExpr);
```

Description: Creates and returns an **ARITHMETIC_EXPRESSION** where the `leftExpr` is multiplied with the `rightExpr`.

Statements Table API

Overview: The Statements Table API describes the methods available to extract information related to statements.

- [getAllStatements]
- [getStatementFromIndex]
- [getStatementsForConstants]
- [getStatementsPatternMatching]
- [insertIntoStatementTable]

```
STATEMENT_LIST getAllStatements(DESIGN_ENT_STMT_NAME stmtType);
```

Description: Returns a **STATEMENT_LIST** of all the statements in the Statements Table.

```
STATEMENT getStatementFromIndex(INTEGER index);
```

Description: Returns the **STATEMENT** with the corresponding `index`.

```
STATEMENT_LIST getStatementsForConstant(INTEGER constant);
```

Description: Returns a **STATEMENT_LIST** with all the statements that contains `constant`.

```
STATEMENT_LIST getStatementsPatternMatching(NODE astNode, BOOLEAN allowBefore, BOOLEAN allowAfter, DESIGN_ENT_STMT_NAME stmtType);
```

Description: // TODO

```
VOID insertIntoStatementTable(STATEMENT statement, INTEGER lineNumber);
```

Description: Inserts a **STATEMENT** `statement` with is corresponding `lineNumber` into the Statements Table.

VarTable API

Overview: The VarTable API describes the methods available to extract information related to variables in the processed SIMPLE program.

- [getAllVariables]
- [getIndexFromVariable]
- [getVariableFromIndex]
- [insertIntoVariableTable]

```
VARIABLE_LIST getAllVariables();
```

Description: Returns a **VARIABLE_LIST** of all variables stored in the VarTable.

```
INTEGER getIndexFromVariable(VARIABLE var);
```

Description: Returns the **INTEGER** key of `var` in the VarTable.

```
VARIABLE getVariableFromIndex(INTEGER index);
```

Description: Returns the **VARIABLE** with `index` as its key in the VarTable. If no there is no such `index`, the function throws an **INVALID_INDEX_ERROR**.

```
INTEGER insertIntoVariableTable(VARIABLE var);
```

Description: Inserts the **VARIABLE** `var` into VarTable. Returns the index that `var` is stored at in the VarTable.