**Приложение Д**

**Исходный код**

**<Launcher.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import org.antlr.runtime.ANTLRFileStream;

import org.antlr.runtime.RecognitionException;

import proto.core.Compiler;

import proto.core.TinyCompiler;

import proto.generator.Generator;

import proto.generator.PromelaGenerator;

public class Launcher {

public static final Map<String, Generator> GENERATORS = new HashMap<String, Generator>() {{

// default generator

put("promela", new PromelaGenerator());

}};

public void launch(Map<String, String> args) throws IOException, RecognitionException {

// set basic preferences from cli

Generator generator = GENERATORS.get(args.get("target"));

Compiler compiler = new TinyCompiler(generator);

// start work

compiler.compile(new ANTLRFileStream(args.get("source")));

}

public static void usage() {

// incorrect call output

String usage = "Welcome to Proto Compiler 0.0.1! [http://github.com/hisohito/proto-code]\n\n" +"USAGE:\n" +

"\t java -cp proto-0.0.1.jar proto.Launcher [--target=<TARGET>] [--output=<OUTPUT\_FILE>] <SOURCE>\n\n" +

"\t <SOURCE> is an proto source file\n" +

"\t <TARGET> is one of supported targets (promela by default)\n" +

"\t <OUTPUT\_FILE> is an output file name (out.pml by default)\n";

System.out.println(usage);

}

public static void main(String[] args) throws IOException, RecognitionException {

Map<String, String> yeld = new HashMap<String, String>();

for(String arg: args) {

if (arg.matches("--\\w+=\\w+")) {

arg = arg.substring(2);

String[] parts = arg.split("=");

yeld.put(parts[0], parts[1]);

} else {

yeld.put("source", arg);

}

}

if (yeld.get("source") == null) {

usage();

return;

}

if (yeld.get("output") == null) yeld.put("output", "out.pml");

if (yeld.get("target") == null) yeld.put("target", "promela");

new Launcher().launch(yeld);

}

}

**<CallNode.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.antlr;

import org.antlr.runtime.CommonToken;

import org.antlr.runtime.Token;

import org.antlr.runtime.tree.CommonTree;

public class CallNode extends CommonTree {

String classname;

public CallNode(int ttype) {

token = new CommonToken(ttype, "");

}

public CallNode(Token t) {

token = t;

}

public CallNode(int ttype, Token t, String cname) {

token = t;

this.classname = cname;

}

public String toString() {

String name = (token.getText().indexOf(".") != -1) ? token.getText()

.substring(token.getText().indexOf(".") + 1) : token.getText();

return this.classname + "::" + (token != null ? name : "");

}

}

**<ClassNode.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.antlr;

import org.antlr.runtime.CommonToken;

import org.antlr.runtime.Token;

import org.antlr.runtime.tree.CommonTree;

public class ClassNode extends CommonTree {

public ClassNode(int ttype) {

token = new CommonToken(ttype, "");

}

public ClassNode(Token t) {

token = t;

}

public String toString() {

return (token != null ? token.getText() : "");

}

}

**<MethodNode.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.antlr;

import org.antlr.runtime.CommonToken;

import org.antlr.runtime.Token;

import org.antlr.runtime.tree.CommonTree;

public class MethodNode extends CommonTree {

String classname;

public MethodNode(int ttype) {

token = new CommonToken(ttype, "");

}

public MethodNode(Token t) {

token = t;

}

public MethodNode(int ttype, Token t, String cname) {

token = t;

this.classname = cname;

}

public String toString() {

return this.classname + "::" + (token != null ? token.getText() : "");

}

}

**<PrototypeNode.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.antlr;

import org.antlr.runtime.CommonToken;

import org.antlr.runtime.Token;

import org.antlr.runtime.tree.CommonTree;

public class PrototypeNode extends CommonTree {

String classname;

public PrototypeNode(int ttype) {

token = new CommonToken(ttype, "");

}

public PrototypeNode(Token t) {

token = t;

}

public PrototypeNode(int ttype, Token t, String cname) {

token = t;

this.classname = cname;

}

public String toString() {

String name = token.getText();

return this.classname + "::" + (token != null ? name : "");

}

}

**<SpecNode.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.antlr;

import org.antlr.runtime.CommonToken;

import org.antlr.runtime.Token;

import org.antlr.runtime.tree.CommonTree;

public class SpecNode extends CommonTree {

public SpecNode(int ttype) {

token = new CommonToken(ttype, "");

}

public SpecNode(Token t) {

token = t;

}

public String toString() {

String name = token.getText();

return (token != null ? name : "");

}

}

**<StateNode.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.antlr;

import org.antlr.runtime.CommonToken;

import org.antlr.runtime.Token;

import org.antlr.runtime.tree.CommonTree;

public class StateNode extends CommonTree {

String classname;

public StateNode(int ttype) {

token = new CommonToken(ttype, "");

}

public StateNode(Token t) {

token = t;

}

public StateNode(int ttype, Token t, String cname) {

token = t;

this.classname = cname;

}

public String toString() {

return classname + "::" + (token != null ? token.getText() : "");

}

}

**<Compiler.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.core;

import org.antlr.runtime.ANTLRFileStream;

import org.antlr.runtime.RecognitionException;

public interface Compiler {

public String compile(ANTLRFileStream fs) throws RecognitionException;

}

**<AbstractCompiler.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.core;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import org.antlr.runtime.tree.CommonTree;

import proto.antlr.CallNode;

import proto.antlr.PrototypeNode;

import proto.antlr.StateNode;

import proto.generator.Generator;

import proto.ir.Automata;

public abstract class AbstractCompiler implements Compiler {

protected CommonTree mainTree;

protected Generator generator;

public AbstractCompiler(Generator generator) {

this.generator = generator;

}

private void findStates(CommonTree tree, Automata automata) {

if (tree != null) {

for (int i = 0; i < tree.getChildCount(); i++) {

CommonTree current = (CommonTree) tree.getChild(i);

if (current instanceof StateNode) {

automata.addState(current.toString());

}

findStates((CommonTree) tree.getChild(i), automata);

}

}

}

private CommonTree getSubTree(CommonTree tree, String name) {

if (tree != null) {

for (int i = 0; i < tree.getChildCount(); i++) {

String current = tree.getChild(i).toString();

if (current.equals(name)) {

return (CommonTree) tree.getChild(i);

}

CommonTree t = getSubTree((CommonTree) tree.getChild(i), name);

if (t != null)

return t;

}

return null;

}

return null;

}

private void formPaths(CommonTree tree, Automata automata) {

if (tree != null) {

for (int i = 0; i < tree.getChildCount(); i++) {

CommonTree currentNode = (CommonTree) tree.getChild(i);

String current = currentNode.toString();

if (currentNode instanceof CallNode) {

if (!(currentNode.getParent() instanceof StateNode))

continue;

if (automata.hasState(current)) {

// this is call if StateNode -> ad path to automata

CommonTree parent = (CommonTree) currentNode.getParent();

String from = parent.toString();

if (automata.hasState(from)) {

automata.addPath(from, current);

}

} else {

// if it is MethodCall - check if there are StateNodes in his depth

Collection<String> resolve = resolveCall(tree, current, automata);

if (resolve != null) {

// check all CallNode's in method

for (String r : resolve) {

CommonTree fromNode =

(CommonTree) currentNode.getParent();

while (!(fromNode instanceof StateNode)) {

fromNode = (CommonTree)

fromNode.getParent();

}

String from = fromNode.toString();

if (automata.hasState(from)) {

automata.addPath(from, r);

}

}

}

}

} else if (currentNode instanceof PrototypeNode){

formSpec(currentNode, automata);

break;

}

formPaths((CommonTree) tree.getChild(i), automata);

}

}

}

private String formSpec (CommonTree tree, Automata automata){

if (tree != null) {

String result = "";

for (int i = 0; i < tree.getChildCount(); i++) {

CommonTree current = (CommonTree) tree.getChild(i);

result = current.toString()+" ";

for (int j = 0; j < current.getChildCount(); j++) {

result += current.getChild(j).toString() +" ";

if (current.getChild(j).getChildCount() != 0){

result = formSpec((CommonTree) current.getChild(j),automata)+" ";

}

}

automata.addSpec(result);

}

}

return "";

}

private Collection<String> resolveCall(CommonTree tree, String id,

Automata automata) {

List<String> list = new ArrayList<String>();

// find exactly this MethodNode in tree

CommonTree methodNode = this.getSubTree(this.mainTree, id);

// if there is no childs - :`-(

if (methodNode == null || methodNode.getChildCount() == 0) {

return null;

} else {

// for all node childs:

// 1. if it is CallNode to StateNode - add to list

// 2. if it is CallNode to MethodNode - add to list recursive work result

// return list

for (Object child : methodNode.getChildren()) {

String callName = child.toString();

if (automata.hasState(callName)) {

list.add(callName);

} else {

List<String> addList = (List<String>) resolveCall(tree, callName, automata);

list.addAll(addList);

}

}

}

return list;

}

protected Automata compile0(CommonTree tree) {

mainTree = tree;

Automata automata = new Automata();

findStates(tree, automata);

formPaths(tree, automata);

automata.print();

return automata;

}

}

**<TinyCompiler.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.core;

import org.antlr.runtime.ANTLRFileStream;

import org.antlr.runtime.RecognitionException;

import org.antlr.runtime.TokenRewriteStream;

import org.antlr.runtime.tree.CommonTree;

import proto.antlr.ProtoLexer;

import proto.antlr.ProtoParser;

import proto.generator.Generator;

import proto.ir.Automata;

public class TinyCompiler extends AbstractCompiler implements Compiler {

public TinyCompiler(Generator generator) {

super(generator);

}

public static void printTree(CommonTree t, int indent) {

if (t != null) {

StringBuffer sb = new StringBuffer(indent);

if (t.getParent() == null) {

System.out.println(sb.toString() + t.getText());

}

for (int i = 0; i < indent; i++)

sb = sb.append(" ");

for (int i = 0; i < t.getChildCount(); i++) {

System.out.println(sb.toString() + t.getChild(i));

printTree((CommonTree) t.getChild(i), indent + 1);

}

}

}

public String compile(ANTLRFileStream fs) throws RecognitionException {

ProtoParser parser = new ProtoParser(new TokenRewriteStream(new ProtoLexer(fs)));

CommonTree tree = (CommonTree) parser.program().getTree();

printTree(tree, 0);

Automata ir = compile0(tree);

return generator.generate(ir);

}

}

**<Generator.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.generator;

import proto.ir.Automata;

public interface Generator {

public String generate(Automata automata);

}

**<PromelaGenerator.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.generator;

import java.io.DataInputStream;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Collection;

import java.util.Collections;

import java.util.HashSet;

import java.util.Set;

import proto.ir.Automata;

public class PromelaGenerator implements Generator {

private Collection<String> generateSpecs(Automata automata) {

Set<String> allSpec = new HashSet<String>();

Collection<String> specs = automata.specs();

for (String spec : specs) {

// replace words

spec = spec.replaceAll("finally", "<>")

.replaceAll("globally", "[]")

.replaceAll("until", "U")

.replaceAll("release", "R");

spec = spec.replaceAll("not", "!")

.replaceAll("and", "&&")

.replaceAll("or", "||")

.replaceAll("if", "->");

String tmp[] = spec.split(" ");

StringBuilder oneSpec = new StringBuilder();

for (String string : tmp) {

if (!string.equals("R") && !string.equals("->")

&& !string.equals("||") && !string.equals("&&")

&& !string.equals("U") && !string.equals("!")

&& !string.equals("{") && !string.equals("}")

&& !string.equals("(") && !string.equals(")")

&& !string.equalsIgnoreCase("[]")

&& !string.equalsIgnoreCase("<>")) {

if (automata.hasState(string)) {

// insert state 'stateA1' expression

oneSpec.append("stateA1 == "+ string.toUpperCase().replace(':', '\_'));

} else {

System.err.println("ERROR: Wrong specification description!");

break;

}

} else {

oneSpec.append(string);

}

}

// send string to spin

oneSpec.insert(0, "!(");

oneSpec.append(")");

Runtime rt = Runtime.getRuntime();

Process proc;

try {

System.out.println("spin.exe -f \"" + oneSpec.toString()+"\"");

proc = rt.exec("spin.exe -f \"" + oneSpec.toString()+"\"");

int exitVal = proc.waitFor();

StringBuilder out = new StringBuilder();

DataInputStream din = new DataInputStream(proc.getInputStream());

String temp = "";

try {

while ((temp = din.readLine()) != null) {

out.append(temp);

}

allSpec.add(out.toString());

} catch (IOException e) {

System.exit(0);

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

return Collections.unmodifiableSet(allSpec);

}

public String generate(Automata automata) {

StringBuilder output = new StringBuilder();

output.append("int stateA1;\nint lastEvent;\n");

int index = 0;

String entryPoint = "";

// states info defines

for (String state : automata.states()) {

state = state.replace(':', '\_');

output.append("#define " + state.toUpperCase() + " " + index + "\n");

index++;

if (state.indexOf("main") != -1) {

entryPoint = state;

}

}

// main function begins here

output.append("\ninline A1() {\n");

// body start

output.append("\tstateA1 = "+ entryPoint.replace(':', '\_').toUpperCase() + ";");

output.append("\n\tdo\n");

for (String state : automata.states()) {

output.append("\t\t:: ( stateA1 == "+ state.replace(':', '\_').toUpperCase() + " ) ->\n");

if (automata.paths(state) != null) {

// printf for log

output.append("\t\t\tprintf(\"" + state + "\");\n");

output.append("\t\t\tif\n");

for (String path : automata.paths(state)) {

output.append("\t\t\t\t::stateA1 = "+ path.replace(':', '\_').toUpperCase() + ";\n");

}

output.append("\t\t\tfi;\n");

} else {

output.append("\t\t\tprintf(\"" + state + "\");\n");

if (state.indexOf("finalize") != -1){

output.append("\t\t\tbreak;\n");

}

}

}

// body end

output.append("\tod;\n}\n");

output.append("proctype Model() {\n");

output.append("\tA1();\n}\n");

output.append("init {\n\trun Model();\n}\n");

// spec time

Collection<String> ltlSpec = generateSpecs(automata);

// for each generated specification - create file

String defaultFilename = "C://Users//Jill//diploma//proto-code//target//protoOutput";

int number = 0;

for (String string : ltlSpec) {

File promela = new File(defaultFilename+(number++)+".pml");

try {

promela.createNewFile();

FileWriter fw = new FileWriter(promela);

fw.write(output.toString()+string);

fw.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

if (ltlSpec.isEmpty()){

File promela = new File(defaultFilename+".pml");

try {

promela.createNewFile();

FileWriter fw = new FileWriter(promela);

fw.write(output.toString());

fw.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

return "Success created "+number+" files";

}

}

**<Automata.java>**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Proto Project

//

// Copyright (c) 2012 Julia Shatilina <julia.shatilina@gmail.com>

//

// Licensed under the Apache License, Version 2.0 (the "License");

// You may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package proto.ir;

import java.util.Collection;

import java.util.Collections;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Map;

import java.util.Set;

public final class Automata {

private Set<String> states;

private Map<String, Set<String>> paths;

private Set<String> LTLSpec;

public Automata() {

states = new HashSet<String>();

paths = new HashMap<String, Set<String>>();

LTLSpec = new HashSet<String>();

}

public void addState(String id) {

states.add(id);

}

public boolean hasState(String id) {

return states.contains(id);

}

public Collection<String> states() {

return Collections.unmodifiableSet(states);

}

public void addPath(String fromId, String toId) {

Set<String> path = paths.get(fromId);

if (path == null) {

path = new HashSet<String>();

path.add(toId);

paths.put(fromId, path);

} else {

path.add(toId);

}

}

public Collection<String> paths(String state) {

return paths.get(state);

}

public boolean hasPath(String fromId, String toId) {

return paths.get(fromId).contains(toId);

}

public void addSpec(String id) {

LTLSpec.add(id);

}

public Collection<String> specs() {

return Collections.unmodifiableSet(LTLSpec);

}

public void print() {

for (String state : this.states()) {

System.out.println(state);

if (this.paths(state) != null) {

for (String path : this.paths(state)) {

System.out.println("\t" + path);

}

}

}

}

}

**<ProtoParser.java>**

public static class if\_operator\_return extends ParserRuleReturnScope {

Object tree;blic Object getTree() { return tree; }

};

// $ANTLR start "if\_operator"

// proto\\antlr\\Proto.g:280:1: if\_operator : 'if' '(' big\_expression ')' operator ( 'else' operator )? -> ( big\_expression )? ( operator )\* ;

public final ProtoParser.if\_operator\_return if\_operator() throws RecognitionException {

ProtoParser.if\_operator\_return retval = new ProtoParser.if\_operator\_return();retval.start = input.LT(1);

Object root\_0 = null; Token string\_literal184=null;

Token char\_literal185=null; Token char\_literal187=null;

Token string\_literal189=null;ProtoParser.big\_expression\_return big\_expression186 = null;ProtoParser.operator\_return operator188 = null;ProtoParser.operator\_return operator190 = null;

Object string\_literal184\_tree=null;Object char\_literal185\_tree=null;

Object char\_literal187\_tree=null;Object string\_literal189\_tree=null;

RewriteRuleTokenStream stream\_55=new RewriteRuleTokenStream(adaptor,"token 55");

RewriteRuleTokenStream stream\_17=new RewriteRuleTokenStream(adaptor,"token 17");

RewriteRuleTokenStream stream\_16=new RewriteRuleTokenStream(adaptor,"token 16");

RewriteRuleTokenStream stream\_54=new RewriteRuleTokenStream(adaptor,"token 54");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

RewriteRuleSubtreeStream stream\_operator=new RewriteRuleSubtreeStream(adaptor,"rule operator");

try {// proto\\antlr\\Proto.g:281:5: ( 'if' '(' big\_expression ')' operator ( 'else' operator )? -> ( big\_expression )? ( operator )\* )

// proto\\antlr\\Proto.g:281:7: 'if' '(' big\_expression ')' operator ( 'else' operator )?

{ string\_literal184=(Token)match(input,54,FOLLOW\_54\_in\_if\_operator2078);

stream\_54.add(string\_literal184);

char\_literal185=(Token)match(input,16,FOLLOW\_16\_in\_if\_operator2080);

stream\_16.add(char\_literal185);

pushFollow(FOLLOW\_big\_expression\_in\_if\_operator2082);

big\_expression186=big\_expression(); state.\_fsp--;

stream\_big\_expression.add(big\_expression186.getTree());

char\_literal187=(Token)match(input,17,FOLLOW\_17\_in\_if\_operator2084);

stream\_17.add(char\_literal187);

pushFollow(FOLLOW\_operator\_in\_if\_operator2086);

operator188=operator(); state.\_fsp--;

stream\_operator.add(operator188.getTree());

// proto\\antlr\\Proto.g:281:44: ( 'else' operator )?

int alt41=2; int LA41\_0 = input.LA(1);

if ( (LA41\_0==55) ) { alt41=1; }

switch (alt41) {

case 1 : // proto\\antlr\\Proto.g:281:46: 'else' operator

{ string\_literal189=(Token)match(input,55,FOLLOW\_55\_in\_if\_operator2090);

stream\_55.add(string\_literal189);

pushFollow(FOLLOW\_operator\_in\_if\_operator2092);

operator190=operator(); state.\_fsp--;

stream\_operator.add(operator190.getTree());

}

break;

}

// AST REWRITE

// elements: big\_expression, operator

// token labels:

// rule labels: retval

// token list labels:

// rule list labels:

// wildcard labels:

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 282:5: -> ( big\_expression )? ( operator )\*

{ // proto\\antlr\\Proto.g:282:8: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

} stream\_big\_expression.reset();

// proto\\antlr\\Proto.g:282:24: ( operator )\*

while ( stream\_operator.hasNext() ) {

adaptor.addChild(root\_0, stream\_operator.nextTree());

} stream\_operator.reset();

} retval.tree = root\_0;

}

retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) {

reportError(re);

recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

// $ANTLR end "if\_operator"

public static class for\_operator\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

// $ANTLR start "for\_operator"

// proto\\antlr\\Proto.g:285:1: for\_operator : 'for' '(' assignment ';' big\_expression ';' assignment ')' operator -> assignment ( big\_expression )? assignment operator ;

public final ProtoParser.for\_operator\_return for\_operator() throws RecognitionException {

ProtoParser.for\_operator\_return retval = new ProtoParser.for\_operator\_return();

retval.start = input.LT(1); Object root\_0 = null;

Token string\_literal191=null; Token char\_literal192=null;

Token char\_literal194=null; Token char\_literal196=null;

Token char\_literal198=null;ProtoParser.assignment\_return assignment193 = null;

ProtoParser.big\_expression\_return big\_expression195 = null;

ProtoParser.assignment\_return assignment197 = null;

ProtoParser.operator\_return operator199 = null;

Object string\_literal191\_tree=null;

Object char\_literal192\_tree=null;

Object char\_literal194\_tree=null;

Object char\_literal196\_tree=null;

Object char\_literal198\_tree=null;

RewriteRuleTokenStream stream\_20=new RewriteRuleTokenStream(adaptor,"token 20");

RewriteRuleTokenStream stream\_56=new RewriteRuleTokenStream(adaptor,"token 56");

RewriteRuleTokenStream stream\_17=new RewriteRuleTokenStream(adaptor,"token 17");

RewriteRuleTokenStream stream\_16=new RewriteRuleTokenStream(adaptor,"token 16");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

RewriteRuleSubtreeStream stream\_assignment=new RewriteRuleSubtreeStream(adaptor,"rule assignment");

RewriteRuleSubtreeStream stream\_operator=new RewriteRuleSubtreeStream(adaptor,"rule operator");

try {

// proto\\antlr\\Proto.g:286:5: ( 'for' '(' assignment ';' big\_expression ';' assignment ')' operator -> assignment ( big\_expression )? assignment operator )

// proto\\antlr\\Proto.g:286:7: 'for' '(' assignment ';' big\_expression ';' assignment ')' operator

{

string\_literal191=(Token)match(input,56,FOLLOW\_56\_in\_for\_operator2132);

stream\_56.add(string\_literal191);

char\_literal192=(Token)match(input,16,FOLLOW\_16\_in\_for\_operator2134);

stream\_16.add(char\_literal192);

pushFollow(FOLLOW\_assignment\_in\_for\_operator2136);

assignment193=assignment(); state.\_fsp--;

stream\_assignment.add(assignment193.getTree());

char\_literal194=(Token)match(input,20,FOLLOW\_20\_in\_for\_operator2138);

stream\_20.add(char\_literal194);

pushFollow(FOLLOW\_big\_expression\_in\_for\_operator2140);

big\_expression195=big\_expression(); state.\_fsp--;

stream\_big\_expression.add(big\_expression195.getTree());

char\_literal196=(Token)match(input,20,FOLLOW\_20\_in\_for\_operator2142);

stream\_20.add(char\_literal196);

pushFollow(FOLLOW\_assignment\_in\_for\_operator2145);

assignment197=assignment(); state.\_fsp--;

stream\_assignment.add(assignment197.getTree());

char\_literal198=(Token)match(input,17,FOLLOW\_17\_in\_for\_operator2147);

stream\_17.add(char\_literal198);

pushFollow(FOLLOW\_operator\_in\_for\_operator2149);

operator199=operator();

state.\_fsp--;stream\_operator.add(operator199.getTree());

// AST REWRITE

// elements: assignment, operator, big\_expression, assignment

// token labels:

// rule labels: retval

// token list labels:

// rule list labels:

// wildcard labels:

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 287:5: -> assignment ( big\_expression )? assignment operator

{ adaptor.addChild(root\_0, stream\_assignment.nextTree());

// proto\\antlr\\Proto.g:287:19: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

}

stream\_big\_expression.reset();

adaptor.addChild(root\_0, stream\_assignment.nextTree());

adaptor.addChild(root\_0, stream\_operator.nextTree());

} retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

} catch (RecognitionException re) { reportError(re);

recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { } return retval;

}

// $ANTLR end "for\_operator"

public static class while\_operator\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

// $ANTLR start "while\_operator"

// proto\\antlr\\Proto.g:290:1: while\_operator : 'while' '(' big\_expression ')' operator -> ( big\_expression )? operator ;

public final ProtoParser.while\_operator\_return while\_operator() throws RecognitionException {

ProtoParser.while\_operator\_return retval = new ProtoParser.while\_operator\_return();

retval.start = input.LT(1); Object root\_0 = null;

Token string\_literal200=null; Token char\_literal201=null;Token char\_literal203=null;ProtoParser.big\_expression\_return big\_expression202 = null;

ProtoParser.operator\_return operator204 = null;

Object string\_literal200\_tree=null; Object char\_literal201\_tree=null;

Object char\_literal203\_tree=null;

RewriteRuleTokenStream stream\_57=new RewriteRuleTokenStream(adaptor,"token 57");

RewriteRuleTokenStream stream\_17=new RewriteRuleTokenStream(adaptor,"token 17");

RewriteRuleTokenStream stream\_16=new RewriteRuleTokenStream(adaptor,"token 16");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

RewriteRuleSubtreeStream stream\_operator=new RewriteRuleSubtreeStream(adaptor,"rule operator");

try {

// proto\\antlr\\Proto.g:291:5: ( 'while' '(' big\_expression ')' operator -> ( big\_expression )? operator )

// proto\\antlr\\Proto.g:291:7: 'while' '(' big\_expression ')' operator

{

tring\_literal200=(Token)match(input,57,FOLLOW\_57\_in\_while\_operator2186);

stream\_57.add(string\_literal200);

char\_literal201=(Token)match(input,16,FOLLOW\_16\_in\_while\_operator2188);

stream\_16.add(char\_literal201);

pushFollow(FOLLOW\_big\_expression\_in\_while\_operator2190);

big\_expression202=big\_expression(); state.\_fsp--;

stream\_big\_expression.add(big\_expression202.getTree());

char\_literal203=(Token)match(input,17,FOLLOW\_17\_in\_while\_operator2192);

stream\_17.add(char\_literal203);

pushFollow(FOLLOW\_operator\_in\_while\_operator2194);

operator204=operator(); state.\_fsp--;

stream\_operator.add(operator204.getTree());

// AST REWRITE

// elements: big\_expression, operator

// token labels:

// rule labels: retval

// token list labels:

// rule list labels:

// wildcard labels:

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 292:5: -> ( big\_expression )? operator

{

// proto\\antlr\\Proto.g:292:8: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

}

stream\_big\_expression.reset();

adaptor.addChild(root\_0, stream\_operator.nextTree());

} retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) { reportError(re);recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

} finally { }

return retval;

}

// $ANTLR end "while\_operator"

public static class do\_operator\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

// $ANTLR start "do\_operator"

// proto\\antlr\\Proto.g:295:1: do\_operator : 'do' '{' operator '}' 'while' '(' big\_expression ')' ';' -> operator ( big\_expression )? ;

public final ProtoParser.do\_operator\_return do\_operator() throws RecognitionException {

ProtoParser.do\_operator\_return retval = new ProtoParser.do\_operator\_return(); retval.start = input.LT(1);

Object root\_0 = null; Token string\_literal205=null;

Token char\_literal206=null; Token char\_literal208=null;

Token string\_literal209=null; Token char\_literal210=null;

Token char\_literal212=null; Token char\_literal213=null;

ProtoParser.operator\_return operator207 = null;

ProtoParser.big\_expression\_return big\_expression211 = null;

Object string\_literal205\_tree=null;

Object char\_literal206\_tree=null; Object char\_literal208\_tree=null;

Object string\_literal209\_tree=null; Object char\_literal210\_tree=null;

Object char\_literal212\_tree=null; Object char\_literal213\_tree=null;

RewriteRuleTokenStream stream\_20=new RewriteRuleTokenStream(adaptor,"token 20");

RewriteRuleTokenStream stream\_58=new RewriteRuleTokenStream(adaptor,"token 58");

RewriteRuleTokenStream stream\_57=new RewriteRuleTokenStream(adaptor,"token 57");

RewriteRuleTokenStream stream\_19=new RewriteRuleTokenStream(adaptor,"token 19");

RewriteRuleTokenStream stream\_17=new RewriteRuleTokenStream(adaptor,"token 17");

RewriteRuleTokenStream stream\_18=new RewriteRuleTokenStream(adaptor,"token 18");

RewriteRuleTokenStream stream\_16=new RewriteRuleTokenStream(adaptor,"token 16");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

RewriteRuleSubtreeStream stream\_operator=new RewriteRuleSubtreeStream(adaptor,"rule operator");

try {

// proto\\antlr\\Proto.g:296:5: ( 'do' '{' operator '}' 'while' '(' big\_expression ')' ';' -> operator ( big\_expression )? )

// proto\\antlr\\Proto.g:296:7: 'do' '{' operator '}' 'while' '(' big\_expression ')' ';'

{

string\_literal205=(Token)match(input,58,FOLLOW\_58\_in\_do\_operator2227);

stream\_58.add(string\_literal205);

char\_literal206=(Token)match(input,18,FOLLOW\_18\_in\_do\_operator2229);

stream\_18.add(char\_literal206);

pushFollow(FOLLOW\_operator\_in\_do\_operator2231);

operator207=operator(); state.\_fsp--;

stream\_operator.add(operator207.getTree());

char\_literal208=(Token)match(input,19,FOLLOW\_19\_in\_do\_operator2233);

stream\_19.add(char\_literal208);

string\_literal209=(Token)match(input,57,FOLLOW\_57\_in\_do\_operator2235);

stream\_57.add(string\_literal209);

char\_literal210=(Token)match(input,16,FOLLOW\_16\_in\_do\_operator2237);

stream\_16.add(char\_literal210);

pushFollow(FOLLOW\_big\_expression\_in\_do\_operator2239);

big\_expression211=big\_expression(); state.\_fsp--;

stream\_big\_expression.add(big\_expression211.getTree());

char\_literal212=(Token)match(input,17,FOLLOW\_17\_in\_do\_operator2241);

stream\_17.add(char\_literal212);

char\_literal213=(Token)match(input,20,FOLLOW\_20\_in\_do\_operator2243);

stream\_20.add(char\_literal213);

// AST REWRITE

// elements: operator, big\_expression

// token labels:

// rule labels: retval

// token list labels:

// rule list labels:

// wildcard labels:

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 297:5: -> operator ( big\_expression )?

{ adaptor.addChild(root\_0, stream\_operator.nextTree());

// proto\\antlr\\Proto.g:297:17: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

}

stream\_big\_expression.reset();

} retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) { reportError(re);

recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

} finally { }

return retval;

}

public final ProtoParser.call\_return call() throws RecognitionException {

ProtoParser.call\_return retval = new ProtoParser.call\_return();

retval.start = input.LT(1); Object root\_0 = null; Token i=null;

Token char\_literal181=null; Token char\_literal183=null;

ProtoParser.parameters\_return parameters182 = null;

Object i\_tree=null; Object char\_literal181\_tree=null;

Object char\_literal183\_tree=null;

RewriteRuleTokenStream stream\_17=new RewriteRuleTokenStream(adaptor,"token 17");

RewriteRuleTokenStream stream\_ID=new RewriteRuleTokenStream(adaptor,"token ID");

RewriteRuleTokenStream stream\_16=new RewriteRuleTokenStream(adaptor,"token 16");

RewriteRuleSubtreeStream stream\_parameters=new RewriteRuleSubtreeStream(adaptor,"rule parameters");

try {// proto\\antlr\\Proto.g:262:5: (i= ID '(' parameters ')' -> ^( ID[$ID, scp] ( parameters )? ) )// proto\\antlr\\Proto.g:262:7: i= ID '(' parameters ')'

{

i=(Token)match(input,ID,FOLLOW\_ID\_in\_call2023);stream\_ID.add(i);

char\_literal181=(Token)match(input,16,FOLLOW\_16\_in\_call2025);

stream\_16.add(char\_literal181);

pushFollow(FOLLOW\_parameters\_in\_call2027);

parameters182=parameters(); state.\_fsp--;

stream\_parameters.add(parameters182.getTree());

char\_literal183=(Token)match(input,17,FOLLOW\_17\_in\_call2029);

stream\_17.add(char\_literal183);

if((i!=null?i.getText():null).indexOf(".") != -1) { // if . in call

for(int j=scope.size()-1; j>=0; j--) { if(!scope.get(j).isEmpty()) { String id = scope.get(j).get((i!=null?i.getText():null).substring(0, (i!=null?i.getText():null).indexOf(".")));

if(id != null) {scp=id; break;} }

}

} else { scp = scope.peek().get("self");}

// AST REWRITE

// elements: parameters, ID

// token labels:

// rule labels: retval

// token list labels:

// rule list labels:

// wildcard labels:

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 277:5: -> ^( ID[$ID, scp] ( parameters )? )

{ // proto\\antlr\\Proto.g:277:8: ^( ID[$ID, scp] ( parameters )? )

{

Object root\_1 = (Object)adaptor.nil();

root\_1 = (Object)adaptor.becomeRoot(new CallNode(ID, i, scp), root\_1);

// proto\\antlr\\Proto.g:277:34: ( parameters )?

if ( stream\_parameters.hasNext() ) {

adaptor.addChild(root\_1, stream\_parameters.nextTree());

}

stream\_parameters.reset();

adaptor.addChild(root\_0, root\_1);

}

}

retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

} catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

} // $ANTLR end "call"

public static class call\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

public static class simple\_expression\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

public static class multiplier\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

public static class summand\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

public static class relation\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

public final ProtoParser.return\_operator\_return return\_operator() throws RecognitionException {ProtoParser.return\_operator\_return retval = new ProtoParser.return\_operator\_return();

retval.start = input.LT(1); Object root\_0 = null;

Token string\_literal120=null; Token char\_literal122=null;

ProtoParser.big\_expression\_return big\_expression121 = null;

Object string\_literal120\_tree=null; Object char\_literal122\_tree=null;

RewriteRuleTokenStream stream\_20=new RewriteRuleTokenStream(adaptor,"token 20");

RewriteRuleTokenStream stream\_42=new RewriteRuleTokenStream(adaptor,"token 42");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

try {// proto\\antlr\\Proto.g:200:5: ( 'return' big\_expression ';' -> ( big\_expression )? )// proto\\antlr\\Proto.g:200:7: 'return' big\_expression ';'

{

string\_literal120=(Token)match(input,42,FOLLOW\_42\_in\_return\_operator1429);

stream\_42.add(string\_literal120);

pushFollow(FOLLOW\_big\_expression\_in\_return\_operator1431);

big\_expression121=big\_expression(); state.\_fsp--;

stream\_big\_expression.add(big\_expression121.getTree());

char\_literal122=(Token)match(input,20,FOLLOW\_20\_in\_return\_operator1433);

stream\_20.add(char\_literal122);

// AST REWRITE

// elements: big\_expression

// token labels:

// rule labels: retval

// token list labels:

// rule list labels:

// wildcard labels:

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 201:5: -> ( big\_expression )?

{ // proto\\antlr\\Proto.g:201:8: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

}

stream\_big\_expression.reset();

} retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

} catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

public final ProtoParser.print\_return print() throws RecognitionException {

ProtoParser.print\_return retval = new ProtoParser.print\_return();

retval.start = input.LT(1); Object root\_0 = null;

Token string\_literal117=null; Token char\_literal119=null;

ProtoParser.big\_expression\_return big\_expression118 = null;

Object string\_literal117\_tree=null;

Object char\_literal119\_tree=null;

RewriteRuleTokenStream stream\_20=new RewriteRuleTokenStream(adaptor,"token 20");

RewriteRuleTokenStream stream\_41=new RewriteRuleTokenStream(adaptor,"token 41");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

try {

// proto\\antlr\\Proto.g:195:5: ( 'print' big\_expression ';' -> ( big\_expression )? )

// proto\\antlr\\Proto.g:195:7: 'print' big\_expression ';'

{

string\_literal117=(Token)match(input,41,FOLLOW\_41\_in\_print1394);

stream\_41.add(string\_literal117);

pushFollow(FOLLOW\_big\_expression\_in\_print1396);

big\_expression118=big\_expression(); state.\_fsp--;

stream\_big\_expression.add(big\_expression118.getTree());

char\_literal119=(Token)match(input,20,FOLLOW\_20\_in\_print1398);

stream\_20.add(char\_literal119);

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 196:5: -> ( big\_expression )?

{// proto\\antlr\\Proto.g:196:8: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

}

stream\_big\_expression.reset();

} retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

} catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

// $ANTLR end "print"

public static class return\_operator\_return extends ParserRuleReturnScope {

Object tree; public Object getTree() { return tree; }

};

public final ProtoParser.die\_return die() throws RecognitionException {

ProtoParser.die\_return retval = new ProtoParser.die\_return();

retval.start = input.LT(1); Object root\_0 = null;

Token string\_literal114=null; Token char\_literal116=null;

ProtoParser.big\_expression\_return big\_expression115 = null;

Object string\_literal114\_tree=null; Object char\_literal116\_tree=null;

RewriteRuleTokenStream stream\_20=new RewriteRuleTokenStream(adaptor,"token 20");

RewriteRuleTokenStream stream\_40=new RewriteRuleTokenStream(adaptor,"token 40");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

try {

// proto\\antlr\\Proto.g:190:5: ( 'die' big\_expression ';' -> ( big\_expression )? )

// proto\\antlr\\Proto.g:190:7: 'die' big\_expression ';'

{

string\_literal114=(Token)match(input,40,FOLLOW\_40\_in\_die1359);

stream\_40.add(string\_literal114);

pushFollow(FOLLOW\_big\_expression\_in\_die1361);

big\_expression115=big\_expression(); state.\_fsp--;

stream\_big\_expression.add(big\_expression115.getTree());

char\_literal116=(Token)match(input,20,FOLLOW\_20\_in\_die1363);

stream\_20.add(char\_literal116);

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 191:5: -> ( big\_expression )?

{// proto\\antlr\\Proto.g:191:8: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

}

stream\_big\_expression.reset();

} retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

} catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

// $ANTLR end "die"

public static class print\_return extends ParserRuleReturnScope {

Object tree;

public Object getTree() { return tree; }

};

public final ProtoParser.buildin\_operator\_return buildin\_operator() throws RecognitionException {

ProtoParser.buildin\_operator\_return retval = new ProtoParser.buildin\_operator\_return();

retval.start = input.LT(1); Object root\_0 = null;

ProtoParser.print\_return print111 = null;

ProtoParser.die\_return die112 = null;

ProtoParser.return\_operator\_return return\_operator113 = null;

try { // proto\\antlr\\Proto.g:184:5: ( print | die | return\_operator )

int alt27=3;

switch ( input.LA(1) ) {

case 41: { alt27=1; } break;

case 40: { alt27=2; } break;

case 42: { alt27=3; } break;

default: NoViableAltException nvae =

new NoViableAltException("", 27, 0, input);

throw nvae;

}

switch (alt27) {

case 1 :// proto\\antlr\\Proto.g:184:7: print

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_print\_in\_buildin\_operator1320);

print111=print(); state.\_fsp--;

adaptor.addChild(root\_0, print111.getTree());

} break;

case 2 : // proto\\antlr\\Proto.g:185:7: die

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_die\_in\_buildin\_operator1329);

die112=die();

state.\_fsp--;

adaptor.addChild(root\_0, die112.getTree());

} break;

case 3 : // proto\\antlr\\Proto.g:186:7: return\_operator

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_return\_operator\_in\_buildin\_operator1338);

return\_operator113=return\_operator();

state.\_fsp--;

adaptor.addChild(root\_0, return\_operator113.getTree());

}

break;

}

retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

// $ANTLR end "buildin\_operator"

public static class die\_return extends ParserRuleReturnScope {

Object tree;

public Object getTree() { return tree; }

};

public final ProtoParser.field\_return field() throws RecognitionException {

ProtoParser.field\_return retval = new ProtoParser.field\_return();

retval.start = input.LT(1);

Object root\_0 = null;

Token i=null;

Token char\_literal95=null;

Token char\_literal97=null;

ProtoParser.type\_return t = null;

ProtoParser.big\_expression\_return big\_expression96 = null;

Object i\_tree=null;

Object char\_literal95\_tree=null;

Object char\_literal97\_tree=null;

RewriteRuleTokenStream stream\_20=new RewriteRuleTokenStream(adaptor,"token 20");

RewriteRuleTokenStream stream\_ID=new RewriteRuleTokenStream(adaptor,"token ID");

RewriteRuleTokenStream stream\_39=new RewriteRuleTokenStream(adaptor,"token 39");

RewriteRuleSubtreeStream stream\_big\_expression=new RewriteRuleSubtreeStream(adaptor,"rule big\_expression");

RewriteRuleSubtreeStream stream\_type=new RewriteRuleSubtreeStream(adaptor,"rule type");

try {// proto\\antlr\\Proto.g:162:5: (t= type i= ID ( '=' big\_expression )? ';' -> ( big\_expression )? )

// proto\\antlr\\Proto.g:162:7: t= type i= ID ( '=' big\_expression )? ';'

{

pushFollow(FOLLOW\_type\_in\_field1124);

t=type(); state.\_fsp--;

stream\_type.add(t.getTree());

i=(Token)match(input,ID,FOLLOW\_ID\_in\_field1128);

stream\_ID.add(i);

scope.peek().put((i!=null?i.getText():null), (t!=null?input.toString(t.start,t.stop):null));

// proto\\antlr\\Proto.g:166:7: ( '=' big\_expression )?

int alt24=2; int LA24\_0 = input.LA(1);

if ( (LA24\_0==39) ) { alt24=1; }

switch (alt24) {

case 1 : // proto\\antlr\\Proto.g:166:9: '=' big\_expression

{

char\_literal95=(Token)match(input,39,FOLLOW\_39\_in\_field1148);

stream\_39.add(char\_literal95);

pushFollow(FOLLOW\_big\_expression\_in\_field1150);

big\_expression96=big\_expression();

state.\_fsp--;

stream\_big\_expression.add(big\_expression96.getTree());

}

break;

}

char\_literal97=(Token)match(input,20,FOLLOW\_20\_in\_field1155);

stream\_20.add(char\_literal97);

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 167:5: -> ( big\_expression )?

{ // proto\\antlr\\Proto.g:167:8: ( big\_expression )?

if ( stream\_big\_expression.hasNext() ) {

adaptor.addChild(root\_0, stream\_big\_expression.nextTree());

}

stream\_big\_expression.reset();

} retval.tree = root\_0;

} retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

} catch (RecognitionException re) {

reportError(re);

recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

// $ANTLR end "field"

public static class operator\_return extends ParserRuleReturnScope {

Object tree;

public Object getTree() { return tree; }

};

public final ProtoParser.body\_return body() throws RecognitionException {

ProtoParser.body\_return retval = new ProtoParser.body\_return();

retval.start = input.LT(1);

Object root\_0 = null; ProtoParser.method\_return method86 = null;

ProtoParser.field\_return field87 = null;

try {

// proto\\antlr\\Proto.g:141:5: ( method | field )

int alt22=2;

alt22 = dfa22.predict(input);

switch (alt22) {

case 1 :

// proto\\antlr\\Proto.g:141:7: method

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_method\_in\_body965);

method86=method();

state.\_fsp--;

adaptor.addChild(root\_0, method86.getTree());

} break;

case 2 :

// proto\\antlr\\Proto.g:142:7: field

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_field\_in\_body974);

field87=field();

state.\_fsp--;

adaptor.addChild(root\_0, field87.getTree());

} break;

}

retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

// $ANTLR end "body"

public static class method\_return extends ParserRuleReturnScope {

Object tree;

public Object getTree() { return tree; }

};

public final ProtoParser.class\_decl\_return class\_decl() throws RecognitionException {

ProtoParser.class\_decl\_return retval = new ProtoParser.class\_decl\_return();

retval.start = input.LT(1);

Object root\_0 = null;

Token i=null;

Token string\_literal73=null;

Token char\_literal74=null;

Token char\_literal76=null;

Token char\_literal77=null;

Token ID78=null;

Token char\_literal79=null;

Token ID80=null;

Token string\_literal81=null;

Token ID82=null;

Token char\_literal83=null;

Token char\_literal85=null;

ProtoParser.arguments\_return arguments75 = null;

ProtoParser.body\_return body84 = null;

Object i\_tree=null;

Object string\_literal73\_tree=null;

Object char\_literal74\_tree=null;

Object char\_literal76\_tree=null;

Object char\_literal77\_tree=null;

Object ID78\_tree=null;

Object char\_literal79\_tree=null;

Object ID80\_tree=null;

Object string\_literal81\_tree=null;

Object ID82\_tree=null;

Object char\_literal83\_tree=null;

Object char\_literal85\_tree=null;

RewriteRuleTokenStream stream\_19=new RewriteRuleTokenStream(adaptor,"token 19");

RewriteRuleTokenStream stream\_17=new RewriteRuleTokenStream(adaptor,"token 17");

RewriteRuleTokenStream stream\_ID=new RewriteRuleTokenStream(adaptor,"token ID");

RewriteRuleTokenStream stream\_18=new RewriteRuleTokenStream(adaptor,"token 18");

RewriteRuleTokenStream stream\_36=new RewriteRuleTokenStream(adaptor,"token 36");

RewriteRuleTokenStream stream\_16=new RewriteRuleTokenStream(adaptor,"token 16");

RewriteRuleTokenStream stream\_27=new RewriteRuleTokenStream(adaptor,"token 27");

RewriteRuleTokenStream stream\_37=new RewriteRuleTokenStream(adaptor,"token 37");

RewriteRuleTokenStream stream\_38=new RewriteRuleTokenStream(adaptor,"token 38");RewriteRuleSubtreeStream stream\_body=new RewriteRuleSubtreeStream(adaptor,"rule body");RewriteRuleSubtreeStream stream\_arguments=new RewriteRuleSubtreeStream(adaptor,"rule arguments");

try {

// proto\\antlr\\Proto.g:127:5: ( 'class' i= ID '(' arguments ')' ( '<' ID ( ',' ID )\* )? ( '<<' ID )? '{' ( body )\* '}' -> ^( ID ( body )\* ) )

// proto\\antlr\\Proto.g:127:7: 'class' i= ID '(' arguments ')' ( '<' ID ( ',' ID )\* )? ( '<<' ID )? '{' ( body )\* '}'

{

string\_literal73=(Token)match(input,36,FOLLOW\_36\_in\_class\_decl845);

stream\_36.add(string\_literal73);

i=(Token)match(input,ID,FOLLOW\_ID\_in\_class\_decl849);

stream\_ID.add(i);

char\_literal74=(Token)match(input,16,FOLLOW\_16\_in\_class\_decl851);

stream\_16.add(char\_literal74);

pushFollow(FOLLOW\_arguments\_in\_class\_decl853);

arguments75=arguments();

state.\_fsp--;

stream\_arguments.add(arguments75.getTree());

char\_literal76=(Token)match(input,17,FOLLOW\_17\_in\_class\_decl855);

stream\_17.add(char\_literal76);

// proto\\antlr\\Proto.g:127:38: ( '<' ID ( ',' ID )\* )?

int alt19=2;

int LA19\_0 = input.LA(1);

if ( (LA19\_0==37) ) {

alt19=1;

}

switch (alt19) {

case 1 :

// proto\\antlr\\Proto.g:127:40: '<' ID ( ',' ID )\*

{

char\_literal77=(Token)match(input,37,FOLLOW\_37\_in\_class\_decl859);

stream\_37.add(char\_literal77);

ID78=(Token)match(input,ID,FOLLOW\_ID\_in\_class\_decl861);

stream\_ID.add(ID78);

// proto\\antlr\\Proto.g:127:47: ( ',' ID )\*

loop18:

do {

int alt18=2;

int LA18\_0 = input.LA(1);

if ( (LA18\_0==27) ) {

alt18=1;

}

switch (alt18) {

case 1 : // proto\\antlr\\Proto.g:127:49: ',' ID

{

char\_literal79=(Token)match(input,27,FOLLOW\_27\_in\_class\_decl865);

stream\_27.add(char\_literal79);

ID80=(Token)match(input,ID,FOLLOW\_ID\_in\_class\_decl867);

stream\_ID.add(ID80);

}

break;

default :

break loop18;

}

} while (true);

} break;

}

// proto\\antlr\\Proto.g:127:62: ( '<<' ID )?

int alt20=2;

int LA20\_0 = input.LA(1);

if ( (LA20\_0==38) ) {

alt20=1;

}

switch (alt20) {

case 1 :

// proto\\antlr\\Proto.g:127:64: '<<' ID

{

string\_literal81=(Token)match(input,38,FOLLOW\_38\_in\_class\_decl877);

stream\_38.add(string\_literal81);

ID82=(Token)match(input,ID,FOLLOW\_ID\_in\_class\_decl879);

stream\_ID.add(ID82);

}

break;

}

char\_literal83=(Token)match(input,18,FOLLOW\_18\_in\_class\_decl884);

stream\_18.add(char\_literal83);

scope.push(new HashMap<String, String>());

scope.peek().put("self", (i!=null?i.getText():null));

// proto\\antlr\\Proto.g:132:8: ( body )\*

loop21:

do {

int alt21=2; int LA21\_0 = input.LA(1);

if ( (LA21\_0==ID||(LA21\_0>=28 && LA21\_0<=33)) ) {

alt21=1;

}

switch (alt21) {

case 1 :

// proto\\antlr\\Proto.g:132:9: body

{

pushFollow(FOLLOW\_body\_in\_class\_decl904);

body84=body();

state.\_fsp--;

stream\_body.add(body84.getTree());

} break;

default : break loop21;

}

} while (true);

scope.pop();

char\_literal85=(Token)match(input,19,FOLLOW\_19\_in\_class\_decl929);

stream\_19.add(char\_literal85);

retval.tree = root\_0;

RewriteRuleSubtreeStream stream\_retval=new RewriteRuleSubtreeStream(adaptor,"rule retval",retval!=null?retval.tree:null);

root\_0 = (Object)adaptor.nil();

// 137:5: -> ^( ID ( body )\* )

{ // proto\\antlr\\Proto.g:137:8: ^( ID ( body )\* )

{

Object root\_1 = (Object)adaptor.nil();

root\_1 = (Object)adaptor.becomeRoot(new ClassNode(stream\_ID.nextToken()), root\_1);

// proto\\antlr\\Proto.g:137:24: ( body )\*

while ( stream\_body.hasNext() ) {

adaptor.addChild(root\_1, stream\_body.nextTree());

}

stream\_body.reset();

adaptor.addChild(root\_0, root\_1);

}

}

retval.tree = root\_0;

}

retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

// $ANTLR end "class\_decl"

public static class body\_return extends ParserRuleReturnScope {

Object tree;

public Object getTree() { return tree; }

};

public static class class\_decl\_return extends ParserRuleReturnScope {

Object tree;

public Object getTree() { return tree; }

};

public final ProtoParser.array\_return array() throws RecognitionException {

ProtoParser.array\_return retval = new ProtoParser.array\_return();

retval.start = input.LT(1);

Object root\_0 = null;

Token char\_literal71=null;

Token char\_literal72=null;

Object char\_literal71\_tree=null;

Object char\_literal72\_tree=null;

try {

// proto\\antlr\\Proto.g:123:5: ( ( '[' ']' )\* )

// proto\\antlr\\Proto.g:123:7: ( '[' ']' )\*

{

root\_0 = (Object)adaptor.nil();

// proto\\antlr\\Proto.g:123:7: ( '[' ']' )\*

loop17:

do {

int alt17=2;

int LA17\_0 = input.LA(1);

if ( (LA17\_0==34) ) {

alt17=1;

}

switch (alt17) {

case 1 :

// proto\\antlr\\Proto.g:123:9: '[' ']'

{

char\_literal71=(Token)match(input,34,FOLLOW\_34\_in\_array819);

char\_literal71\_tree = (Object)adaptor.create(char\_literal71);

adaptor.addChild(root\_0, char\_literal71\_tree);

char\_literal72=(Token)match(input,35,FOLLOW\_35\_in\_array821);

char\_literal72\_tree = (Object)adaptor.create(char\_literal72);

adaptor.addChild(root\_0, char\_literal72\_tree);

}

break;

default :

break loop17;

}

} while (true);

}

retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

public final ProtoParser.method\_decl\_return method\_decl() throws RecognitionException {

ProtoParser.method\_decl\_return retval = new ProtoParser.method\_decl\_return();

retval.start = input.LT(1);

Object root\_0 = null;

Token ID49=null;

Token char\_literal50=null;

Token char\_literal52=null;

Token char\_literal53=null;

ProtoParser.type\_return type48 = null;

ProtoParser.arguments\_return arguments51 = null;

Object ID49\_tree=null;

Object char\_literal50\_tree=null;

Object char\_literal52\_tree=null;

Object char\_literal53\_tree=null;

try { // proto\\antlr\\Proto.g:91:5: ( type ID '(' arguments ')' ';' )

// proto\\antlr\\Proto.g:91:7: type ID '(' arguments ')' ';'

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_type\_in\_method\_decl572);

type48=type();

state.\_fsp--;

adaptor.addChild(root\_0, type48.getTree());

ID49=(Token)match(input,ID,FOLLOW\_ID\_in\_method\_decl574);

ID49\_tree = (Object)adaptor.create(ID49);

adaptor.addChild(root\_0, ID49\_tree);

char\_literal50=(Token)match(input,16,FOLLOW\_16\_in\_method\_decl576);

char\_literal50\_tree = (Object)adaptor.create(char\_literal50);

adaptor.addChild(root\_0, char\_literal50\_tree);

pushFollow(FOLLOW\_arguments\_in\_method\_decl578);

arguments51=arguments();

state.\_fsp--;

adaptor.addChild(root\_0, arguments51.getTree());

char\_literal52=(Token)match(input,17,FOLLOW\_17\_in\_method\_decl580);

char\_literal52\_tree = (Object)adaptor.create(char\_literal52);

adaptor.addChild(root\_0, char\_literal52\_tree);

char\_literal53=(Token)match(input,20,FOLLOW\_20\_in\_method\_decl582);

char\_literal53\_tree = (Object)adaptor.create(char\_literal53);

adaptor.addChild(root\_0, char\_literal53\_tree);

}

retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) {

reportError(re); recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}

public final ProtoParser.statement\_return statement() throws RecognitionException {

ProtoParser.statement\_return retval = new ProtoParser.statement\_return();

retval.start = input.LT(1);

Object root\_0 = null;

ProtoParser.interface\_decl\_return interface\_decl2 = null;

ProtoParser.class\_decl\_return class\_decl3 = null;

ProtoParser.proto\_decl\_return proto\_decl4 = null;

try {// proto\\antlr\\Proto.g:32:5: ( interface\_decl | class\_decl | proto\_decl )

int alt2=3;

switch ( input.LA(1) ) {

case 26:

{ alt2=1; }

break;

case 36:

{ alt2=2; }

break;

case 15:

{ alt2=3; }

break;

default:

NoViableAltException nvae =

new NoViableAltException("", 2, 0, input);

throw nvae;

}

switch (alt2) {

case 1 :

// proto\\antlr\\Proto.g:32:7: interface\_decl

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_interface\_decl\_in\_statement78);

interface\_decl2=interface\_decl();

state.\_fsp--;

adaptor.addChild(root\_0, interface\_decl2.getTree());

}

break;

case 2 :

// proto\\antlr\\Proto.g:33:7: class\_decl

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_class\_decl\_in\_statement86);

class\_decl3=class\_decl();

state.\_fsp--;

adaptor.addChild(root\_0, class\_decl3.getTree());

}

break;

case 3 :

// proto\\antlr\\Proto.g:34:7: proto\_decl

{

root\_0 = (Object)adaptor.nil();

pushFollow(FOLLOW\_proto\_decl\_in\_statement95);

proto\_decl4=proto\_decl();

state.\_fsp--;

adaptor.addChild(root\_0, proto\_decl4.getTree());

}

break;

}

retval.stop = input.LT(-1);

retval.tree = (Object)adaptor.rulePostProcessing(root\_0);

adaptor.setTokenBoundaries(retval.tree, retval.start, retval.stop);

}

catch (RecognitionException re) {

reportError(re);

recover(input,re);

retval.tree = (Object)adaptor.errorNode(input, retval.start, input.LT(-1), re);

}

finally { }

return retval;

}