PSOA-to-TPTP Converter Documentation

Release 1.0.0

Reuben Peter-Paul, Gen Zou

November 07, 2011

CONTENTS

Contents:

CHAPTER

ONE

PROPOSAL FOR PSOA-TO-TPTP CONVERTER

1.1 Introduction

Rule Language is a main type of formal languages for knowledge representation in semntic web technologies. PSOA-RuleML [Bol11a] is a rule language which generalizes POSL [Bol04], F-Logic and W3C's RIF-BLD languages [BoKi10a], [BoKi10b]. IN PSOA-RuleML, the positional-slotted, object-applicative (PSOA) term is introduced as a generalization of the frame term and the class membership term in RIF-BLD, as well as the positional-slotted term in POSL language. The planned two-part implementation of PSOA-RuleML is (1) convert PSOA-RuleML's syntax into TPTP format, and (2) reads and executes the TPTP with Vampire theorem prover [1]. In this project, we are going to implement part (1) of the overarching project.

1.2 Objectives

This project is an important part of PSOA-RuleML implementation. The objectives of the project is as follows:

1. Develop a converter to translate PSOA-RuleML rule language documents into TPTP format documents. The input of the converter is a document conforming to PSOA-RuleML presentation syntax shown in [Bol11a]. The output of the converter is a TPTP-FOF format document.

2. Create some test examples in PSOA-RuleML syntax and their corresponding TPTP format documents for testing the converter. These examples can be further extended to a complete PSOA test suite in the future.

1.3 Methodology

1.3.1 Grammar Specification

In order to develop a converter from PSOA-RuleML rule language into TPTP via *translation* (or transformation). A *program* specified in PSOA-RuleML must be recognized and parsed into a representation that lends itself to such a *transformation*. This generally requires the use of a grammar specification via specialized notation: Backus-Naur Form [2], Extended Backus-Naur Form [3] or more less known Parsing Expression Grammar [4].

[1] http://www.voronkov.com/vampire.cgi
[2] http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form
[3] http://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_Form
[4] http://en.wikipedia.org/wiki/Parsing_expression_grammar

1.3.2 Compiler-compilers

According to [Wp11], a **compiler-compiler** or **compiler generator** is a tool that creates a parser (and lexer), an interpreter/compiler from som form of formal description of a language and machine. The most prevalent form of compiler-compiler is a **parser-generator** whose input is a grammar (like those mentioned above) of a programming language and whose output is a collection of source code for a parser (and lexer) often used as an initial (partial) set of components for a compiler.

[Wp11] also mentions an open problem, that is the "holy grail" of compiler-compilers such that a formal grammar along with a target platforms instruction set may be given as inputs and the result would be a *full* set of compiler components capable of producing executable bytecode for machines implementing the above instruction set.

1.3.3 Abstract Syntax Trees

The result of applying a typical parser generator to a *input program* (or *strings*) is a **concrete syntax tree**, or **parse tree**: an ordered, rooted tree that represents the syntactic structure of the *input* according to some formally specified grammar. In a parse tree, the leaves are labeled by *terminals* of the grammar while the internal nodes are labeled as *non-terminals* of the grammar.

According to [Par07] and [Wp11] typical parser generators associate pieces of executable code (written in a particular target language, e.g. Java, Python, Ruby, etc.). These pieces of code referred to as **actions** or **semantic action routines** are executed when a particular rule of the grammar is applied by the parser. These routines may be used to specify the semantics of the syntactic structure that is analyzed by the parser.

The result of applying the generated parser with **semantic action routines** is an **abstract syntax tree**, or alternatively executable code. In this project we will concern ourselves with **abstract syntax trees** only, such that the *transformation/translation* we intend to perform will be done against an **AST** to produce yet another **AST** that may be used to yield *executable* **TPTP-FOF** strings.

1.4 Tools

1.4.1 RuleML API

The primary purpose of the PSOA RuleML API is to provide abstract syntax classes convenient for manipulating language constructs. Any Java sofware using or processing PSOA RuleML will be able to benefit from this functionality. The API will also provide two native parsers into the abstract syntax: a presentation syntax parser to support simple text editor-based authoring, and a parser for the XML serialisation, intended to support rule interchange. [Ria11]

1.4.2 ANTLR and AntlrWorks

We will be working with ANTLR, a sophisticated parser-generator tool that is popular and used to implement language interpreters and compilers.

ANTLRWorks is a graphical user interface development environment used to facilitate the development of grammars (in **EBNF**) by providing the researcher with tooling for testing/debugging the generated parser, running the parser against various inputs, running rules in isolation against various inputs, visualizing syntax diagrams of the grammar rules, visualizing the **concrete syntax tree** produced by applying the generated parser against inputs.

1.4.3 ANTLR Tree Description Language

As mentioned above parser generators typically support **action routines** to be executed "on-application" of a given rule, ANTLR is no exception. Additionally, it supports a separate mechanism for constructing **trees** using the ->

operator and **ANTLR treed description language**. Using ANTLR's tree description language, a tree is written like this:

```
^( CLASS T ^(VARDEF int i) ^(VARDEF int j) ^(METHOD ...) ...)
```

This notation may also be used to recognize *subtrees*, introducing a higher level of abstraction, more suitable for mananging complexities of translating a PSOA-RuleML to another language such as TPTP-FOF.

Project Proposal

BIBLIOGRAPHY

[Bol11a] Boley H., A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules, RuleML Europe 2011, 194-211

[Bol04] Boley H., POSL: An Integrated Positional-Slotted Language for Semantic Web Knowledge, http://ruleml.org/submission/ruleml-shortation.html

[BoKi10a] Boley H., M. Kifer, A Guide to the Basic Logic Dialect for Rule Interchange on the Web. IEEE Transactions on Knowledge and Data Engineering, 22(11):1593-1608

[BoKi10b] Boley H., M. Kifer, RIF Basic Logic Dialect, http://www.w3.org/TR/rif-bld/

[Wp11] Compiler-Compiler, Wikipedia: The Free Encylopedia, http://en.wikipedia.org/wiki/Parser_generator

[Par07] Parr T., The Definitive ANTLR Reference: Building Domain Specific Languages, 2007, Pragmatic Programmer, USA.

[Ria11] Skype conversation with Alex Riazanov