# PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners

Gen Zou[1] Reuben Peter-Paul[1] Harold Boley[1,2]
Alexandre Riazanov[3]

[1]Faculty of Computer Science,
University of New Brunswick, Fredericton, Canada

[2]National Research Council Canada,
Information and Communications Technologies

[3]Department of Computer Science & Applied Statistics
University of New Brunswick, Saint John, Canada

# Outline

1. Background
2. System Architecture
3. Implementation
   - ANTLR Grammars
   - PSOA-to-TPTP Translation
   - RESTful Web API
4. Summary and Future Work

# Outline

1. Background
2. System Architecture
3. Implementation
   - ANTLR Grammars
   - PSOA-to-TPTP Translation
   - RESTful Web API
4. Summary and Future Work

# Interoperation Source – PSOA RuleML

- Integrates relational and object-oriented modeling
- Generalizes RIF-BLD, F-logic and POSL
- Uses positional-slotted object-applicative (psoa) terms

  General case:

  $o\#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1{->}v_1 \ldots p_k{->}v_k)$

  Special cases:

  $o\#f(t_1 \ldots t_n\ p_1{->}v_1 \ldots p_k{->}v_k)$

  $o\#f(t_1 \ldots t_n)$

  $o\#f(\qquad p_1{->}v_1 \ldots p_k{->}v_k)$

  $o\#f$

# PSOA Example in Presentation Syntax (PS)

```
Document(
    Group (
          Forall ?Hu ?Wi ?Ch ?o ?1 (
              ?o#_family(_child->?Ch) :-
                 And(?o#_family(_husb->?Hu _wife->?Wi)
                     ?1#_kid(?Wi ?Ch))
          )
          _f1#_family(_husb->_Joe _wife->_Sue)
          _k1#_kid(_Sue _Pete)
    )
)
```

# Interoperation Targets

- TPTP's full First-Order Form (TPTP-FOF)
  - Allows arbitrary first-order formulas
  - Syntax and semantics

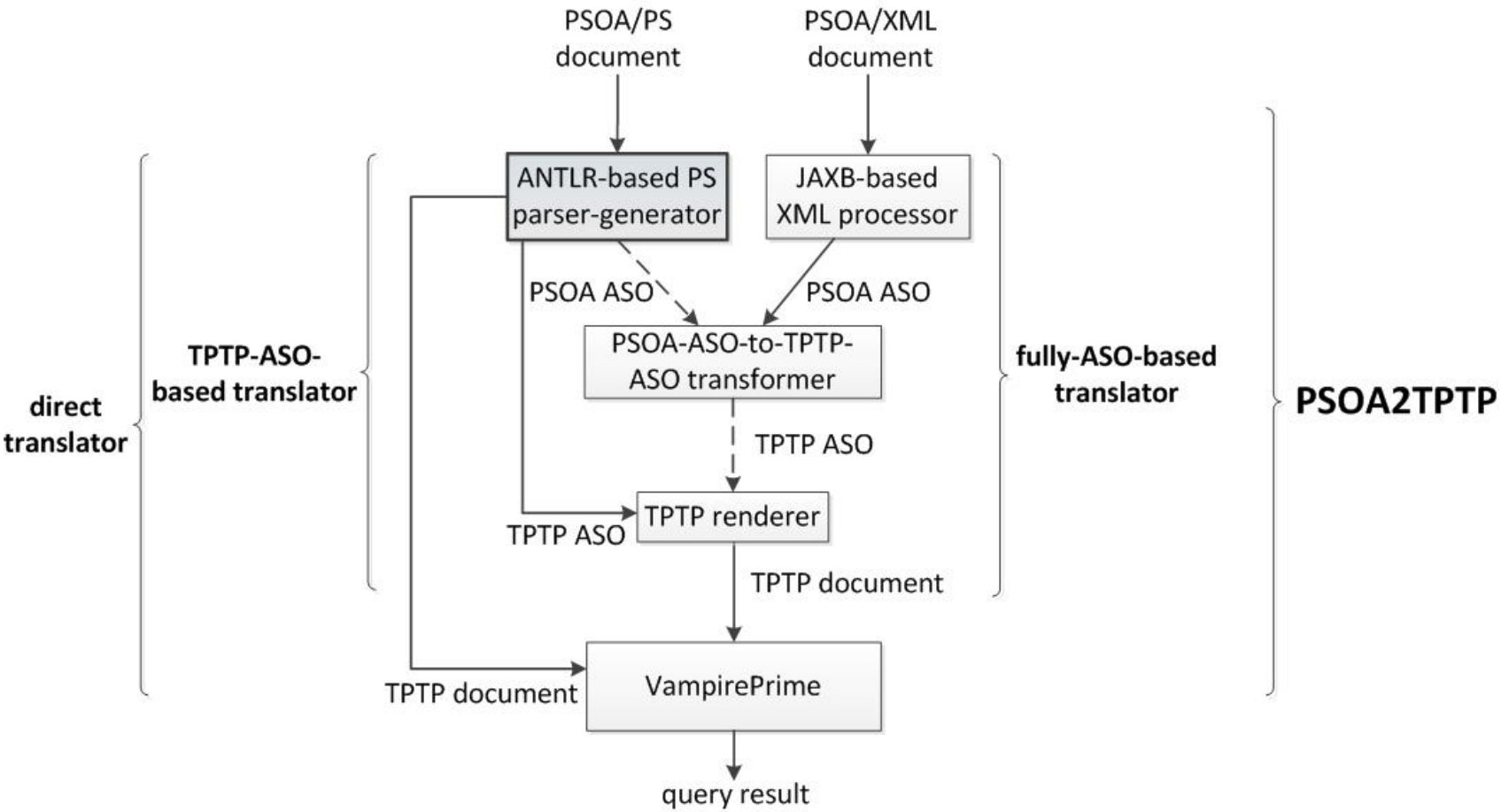
| Syntax | Semantics | Syntax | Semantics |
|---|---|---|---|
| ~p | $\neg p$ | $v_1 = v_2$ | $v_1 = v_2$ |
| $p_1$ & $p_2$ | $p_1 \wedge p_2$ | $v_1$ != $v_2$ | $\neg ( v_1 = v_2 )$ |
| $p_1$ \| $p_2$ | $p_1 \vee p_2$ | ?[$v_1$,...,$v_n$] : p | $\exists\, v_1, \ldots, v_n : p$ |
| $p_1$ => $p_2$ | $p_1 \rightarrow p_2$ | ![$v_1$,...,$v_n$] : p | $\forall\, v_1, \ldots, v_n : p$ |

- VampirePrime
  Open-source TPTP reasoner derived from the high-performance Vampire reasoner

# Outline

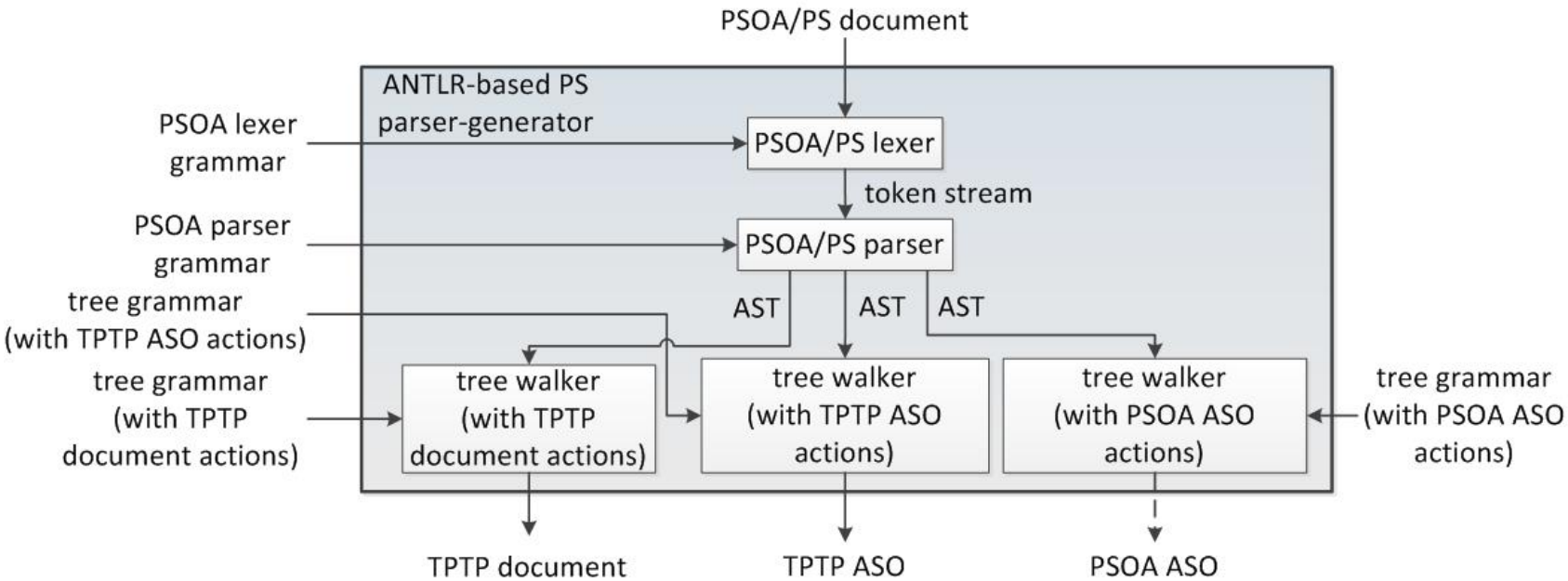
1. Background
2. System Architecture
3. Implementation
   - ANTLR Grammars
   - PSOA-to-TPTP Translation
   - RESTful Web API
4. Summary and Future Work

# Components and Workflow of PSOA2TPTP




ANTLR: ANother Tool for Language Recognition
ASO: Abstract Syntax Object

# Workflow of ANTLR-Generated Components




AST: Abstract Syntax Tree (condensed tree encoding of the input stream)

# Outline

1. Background
2. System Architecture
3. Implementation
   - ANTLR Grammars
   - PSOA-to-TPTP Translation
   - RESTful Web API
4. Summary and Future Work

# Outline

1 Background

2 System Architecture

3 Implementation
- ANTLR Grammars
- PSOA-to-TPTP Translation
- RESTful Web API

4 Summary and Future Work

# Parser Grammar Rewriting

- ANTLR-generated parser uses *LL* (Left-to-right scanning, Leftmost derivation) mechanism
- The original EBNF grammar of PSOA/PS extended with syntactic sugar is non-*LL* and cannot be directly used as the ANTLR parser grammar
- We rewrite it into an *LL*(1) grammar, so that it not only can be accepted by ANTLR but is also more efficient
  - Elimination of left recursion
  - Left factoring

# Parser Grammar Rewriting

- Elimination of left recursion – example

```
psoa :  term '#' term ('(' tuples_and_slots ')')?
     |  term '(' tuples_and_slots ')' ;
term :  const | var | psoa | external_term
```

is rewritten to

```
term :  psoa | non_psoa_term ;
non_psoa_term :  const | var | external_term ;
psoa :  non_psoa_term psoa_rest+ ;
```

# Parser Grammar Rewriting

- Left factoring – example

```
tuples_and_slots: tuple* (term '->' term)*
                | term+ (term '->' term)* ;
```

is rewritten to

```
tuples_and_slots: tuple+ (term '->' term)*
                | term+ ('->' term (term '->' term)*)?
                |
                ;
```

# Outline

1 Background

2 System Architecture

3 Implementation
- ANTLR Grammars
- PSOA-to-TPTP Translation
- RESTful Web API

4 Summary and Future Work

# Normalization – Static Tupribution and Slotribution

Transform composite formulas into a conjunction of elementary constructs

- $o\#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1{->}v_1 \ldots p_k{->}v_k)$
  $=\ \ o\#f()$
  $\&\ o\#Top(t_{1,1} \ldots t_{1,n_1})\ \&\ldots\&\ o\#Top(t_{m,1} \ldots t_{m,n_m})$
  $\&\ o\#Top(p_1{->}v_1)\ \&\ldots\&\ o\#Top(p_k{->}v_k)$

# Translation of Elementary Constructs – Terms

- Constants $\tau_{psoa}(\texttt{\_C}) = \texttt{lC}$ (“low line” C)
- Variables $\tau_{psoa}(\texttt{?v}) = \texttt{Qv}$ (“Question mark” v)
- Tuple terms

  $\tau_{psoa}(\texttt{o\#Top(t}_1 \ldots \texttt{t}_k\texttt{)}) =$
  $\texttt{tupterm(}\tau_{psoa}(\texttt{o}), \tau_{psoa}(\texttt{t}_1) \ldots \tau_{psoa}(\texttt{t}_k)\texttt{)}$
- Slot terms

  $\tau_{psoa}(\texttt{o\#Top(p -> v)}) =$
  $\texttt{sloterm(}\tau_{psoa}(\texttt{o}), \tau_{psoa}(\texttt{p}), \tau_{psoa}(\texttt{v})\texttt{)}$
- Membership terms

  $\tau_{psoa}(\texttt{o\#f()}) = \texttt{member(}\tau_{psoa}(\texttt{o}), \tau_{psoa}(\texttt{f})\texttt{)}$

# Translation of Elementary Constructs – Formulas

- Conjunction

  $\tau_{psoa}(\texttt{And(f}_1 \texttt{ ... f}_n\texttt{)) = (}\tau_{psoa}(\texttt{f}_1\texttt{) \& ... \&}\ \tau_{psoa}(\texttt{f}_n\texttt{))}$
- Implication $\tau_{psoa}(\varphi \texttt{ :- } \psi) \texttt{ = (}\tau_{psoa}(\psi) \texttt{=>} \tau_{psoa}(\varphi)\texttt{)}$
- Existential Quantification

  $\tau_{psoa}(\texttt{Exists ?v}_1\texttt{...?v}_n\texttt{ f) =}$
  $\texttt{(? [}\tau_{psoa}(\texttt{?v}_1)\texttt{...}\tau_{psoa}(\texttt{?v}_n)\texttt{] : }\tau_{psoa}(\texttt{f})\texttt{)}$
- Universal Quantification

  $\tau_{psoa}(\texttt{Forall ?v}_1\texttt{...?v}_n\texttt{ f) =}$
  $\texttt{(! [}\tau_{psoa}(\texttt{?v}_1)\texttt{...}\tau_{psoa}(\texttt{?v}_n)\texttt{] : }\tau_{psoa}(\texttt{f})\texttt{)}$

# Translation of Queries

Use reserved answer predicate `ans` to obtain the bindings for query variable $?v_i$

$$! \ [\tau_{psoa}(?v_1)...\tau_{psoa}(?v_n)] \ : \ (\tau_{psoa}(q) \ :- \ ans("?v_1 \ = \ ", \tau_{psoa}(?v_1), \ ..., \ "?v_n \ = \ ", \tau_{psoa}(?v_n)))$$

# Example – Input

- Input knowledge base

```
Document(
    Group (
          Forall ?Hu ?Wi ?Ch ?o ?1 (
              ?o#_family(_child->?Ch) :-
                 And(?o#_family(_husb->?Hu _wife->?Wi)
                     ?1#_kid(?Wi ?Ch))
          )
          _f1#_family(_husb->_Joe _wife->_Sue)
          _k1#_kid(_Sue _Pete)
    )
)
```

- Query

```
And(_f1#_family(_husb->_Joe _wife->_Sue _child->?Who)
    _k1#_kid(_Sue ?Who))
```

# Example – Normalization

- Normalized knowledge base

```
Document(
    Group (
          Forall ?Hu ?Wi ?Ch ?o ?1 (
              And( ?o#_family()
                   ?o#Top(_child->?Ch)) :-
                  And( And( ?o#_family()
                            ?o#Top(_husb->?Hu)
                            ?o#Top(_wife->?Wi))
                       And( ?1#_kid()
                            ?1#Top(?Wi ?Ch)))
          )
          _f1#_family()   _f1#Top(_husb->_Joe)
                                  _f1#Top(_wife->_Sue)
          _k1#_kid()   _k1#Top(_Sue _Pete)
    )
)
```

# Example – Translation

- TPTP generated for knowledge base

```
fof(ax01,axiom,(
    !  [Q1,Qo,QCh,QWi,QHu] :
      ( ( member(Qo,lfamily)
      & sloterm(Qo,lwife,QWi)
      & sloterm(Qo,lhusb,QHu)
      &  member(Q1,lkid)
      & tupterm(Q1,QWi,QCh) )
    => ( member(Qo,lfamily)
      & sloterm(Qo,lchild,QCh) ) ))).
fof(ax02,axiom,
    ( member(f1,lfamily)
    & sloterm(f1,lwife,lSue)
    & sloterm(lf1,lhusb,lJoe) )).
fof(ax03,axiom,
    ( member(lk1,lkid) & tupterm(k1,lSue,lPete) )).
```

# Example – Translation and Execution

- TPTP generated for query

```
fof(query,theorem,(
     !  [QWho] :
       ( ( member(f1,lfamily)
       & sloterm(f1,lwife,lSue)
       & sloterm(f1,lchild,QWho)
       & sloterm(f1,lhusb,lJoe)
       & member(lk1,lkid)
       & tupterm(lk1,lSue,QWho) )
      => ans("?Who = ",QWho) ) )).
```

- VampirePrime output

```
Proof found.
...
... | «ans»("?Who = ", lPete) ...
```

# Outline

1. Background
2. System Architecture
3. Implementation
   - ANTLR Grammars
   - PSOA-to-TPTP Translation
   - RESTful Web API
4. Summary and Future Work

# RESTful Web API

- PSOATransRun wraps PSOA2TPTP and VampirePrime into two REST services
  - Translation: PSOA2TPTP
  - Execution: VampirePrime
- Requests are sent via HTTP POST method
- Inputs and outputs are JSON-encoded strings
- Available online (documentation and system):

  `http://wiki.ruleml.org/index.php/PSOA_RuleML#PSOATransRun`

  `http://198.164.40.211:8082/psoa2tptp-trans/index.html`

# Outline

1. Background
2. System Architecture
3. Implementation
   - ANTLR Grammars
   - PSOA-to-TPTP Translation
   - RESTful Web API
4. Summary and Future Work

# Summary

- We have implemented a first version of the PSOA2TPTP translator using the ANTLR v3 framework
- Thus provide a semantics-preserving translation from PSOA RuleML to TPTP
- PSOATransRun wraps PSOA2TPTP and VampirePrime into REST services for convenient access

# Future Work

- Extend PSOA2TPTP capability to handle all PSOA RuleML constructs
- Implement the fully-ASO-based translator
- Develop real-world Clinical Intelligence use case and others employing PSOATransRun

# References


Boley, H.
A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules
In Bassiliades, N., Governatori, G., Paschke, A. (eds.)
*RuleML Europe*, vol. 6826 of *LNCS*, pp. 194–211. Springer, 2011.


Zou, G., Peter-Paul, R., Boley, H. and Riazanov, A.
PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners
In: Bikakis, A., Giurca, A. (eds.)
*RuleML 2012*, vol. 7438 of *LNCS*, pp. 264–279. Springer, 2012.


Al Manir, M.S., Riazanov, A., Boley, H. and Baker, C.J.O.
PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes
In: Bikakis, A., Giurca, A. (eds.)
*RuleML 2012*, vol. 7438 of *LNCS*, pp. 280–288. Springer, 2012.