tuProlog (2P)

A Java-based Prolog System for Pervasive Intelligence

Andrea Omicini & Alessandro Ricci
DEIS, Università di Bologna a Cesena

Outline

- Introduction: 2P Objectives & Design
  - 2P technology & tools
- Using 2P from Java
  - 2P API
- Extending 2P: Libraries
- Using Java from 2P: The JavaLibrary

What is 2P

- Java-Based Prolog Virtual Machine
  - minimality & dynamic/open extensibility/configurability
  - bidirectional Java integration
- Java API to use Prolog From Java
  - `alice.tuprolog` package
- 2P Extensions/Tools
  - `alice.tuprologx` package
  - IDEs (`alice.tuprologx.ide.*`)
  - Extensions (libraries, theories)

2P Targets

- As components to be integrated in Java development for *intelligent* application
  - Symbolic Reasoning in a OOP context
  - *"Intelligence + Interaction"*
- Internet Applications
  - as lightweight component for applet, servlet, mobile code, network-enabled devices
- Intelligent Infrastructures
  - Building agent systems
  - ReSpecT Coordination framework & TuCSoN Infrastructure

Getting Started

- 2P is open source
- Download from

  http://lia.deis.unibo.it/research/tuprolog

  http://tuprolog.sourceforge.net
- Download what

  `2P.jar`

  Full package: `2p-X.Y.Z.zip`

  - bin/src/lib/doc

Getting Started: 2P from Java

- From Java

  Using 2P engines in the context of Java applications

  Include the package `2p.jar` in the classpath

  Package to import: `alice.tuprolog.*`

The 2P Tools: IDEs

- Command Line IDE

  `alice.tuprologx.ide.CUIConsole` application

  `java -cp 2p.jar alice.tuprologx.ide.CUIConsole`

- GUI based IDE (version 1.2.1)

  `alice.tuprologx.ide.GUILauncher` application

  `java -cp 2p.jar alice.tuprologx.ide.GUILauncher`

The 2P Tools: Others

- Spawning an agent

  `alice.tuprolog.Agent` application

```
java -cp tuprolog.jar
      alice.tuprolog.Agent PrologTheoryTextFile {Goal}
```

- Examples

  text file `hello.pl` containing

  spawning the agent:

```
java -cp tuprolog.jar
      alice.tuprolog.Agent hello.pl go.
```

2P Main Elements

- Core Inferential Engine

  class Prolog
- Theories

  class Theory
- Libraries

  class Library (abstract)
- Prolog Data Types

  class Term (abstract), Number (abstract), Int/Long/Float/Double (concrete), Struct (concrete), Var (concrete)

2P Data Types

2P Engine Design

- Self-Contained
  - No static links
  - Enabling multiple engines, independently configured (theory, libraries), in the same computational context
- Minimal Interface
  - setting, retrieving, adding theories
  - solving goal
  - load & unload libraries

An example: a theory for expression derivation & evaluation (1)

```
dExpr(T,X,DT):-dTerm(T,X,DT).
dExpr(E+T,X,[DE+DT]):-dExpr(E,X,DE), dTerm(T,X,DT).
dExpr(E-T,X,[DE-DT]):-dExpr(E,X,DE), dTerm(T,X,DT).
dTerm(F,X,DF):-dFactor(F,X,DF).
dTerm(T*F,X,[[DT*F]+[T*DF]]):-dTerm(T,X,DT), dFactor(F,X,DF).
dTerm(T/F,X,[[F*DT]-[T*DF]]/[F*F]):-dTerm(T,X,DT),
dFactor(F,X,DF).
dFactor(-E,X,-DE):-dExpr(E,X,DE).
dFactor([E],X,DE):-dExpr(E,X,DE).
dFactor(N,X,0):-number(N).
dFactor(X,X,1).
dFactor(F^N,X,[[N*[F^M]]*DF]):-M is N-1, dFactor(F,X,DF).
dFactor(sin(E),X,[cos(E)*DE]):-dExpr(E,X,DE).
dFactor(cos(E),X,[-sin(E)*DE]):-dExpr(E,X,DE).
```

(cont)

An example: a theory for expression derivation & evaluation (2)

(cont)

```
evalExpr(T,V,R):-evalTerm(T,V,R).
evalExpr(E+T,V,R):-evalExpr(E,V,R1), evalTerm(T,V,R2), R is R1+R2.
evalExpr(E-T,V,R):-evalExpr(E,V,R1), evalTerm(T,V,R2), R is R1-R2.
evalTerm(F,V,R):-evalFactor(F,V,R).
evalTerm(T*F,V,R):-evalTerm(T,V,R1), evalFactor(F,V,R2), R is R1*R2.
evalTerm(T/F,V,R):-evalTerm(T,V,R1), evalFactor(F,V,R2), R is R1/R2.
evalFactor(-E,V,R):-evalExpr(E,V,R1), R is -R1.
evalFactor([E],V,R):-evalExpr(E,V,R).
evalFactor(N,V,N):-number(N).
evalFactor(x,V,V).
evalFactor(F^E,V,R):-evalFactor(F,V,R1),evalExpr(E,V,R2),R is R1^R2.
evalFactor(sin(E),V,R):-evalExpr(E,V,R1),R is sin(R1).
evalFactor(cos(E),V,R):-evalExpr(E,V,R1),R is cos(R1).
```

Using the theory from Java

2P API: The Engine

Data Types API Example

2P Libraries
(ver. 1.2.1)

- Structure
- Developing a Library
- Default Libraries
- The JavaLibrary

Library Structure

- Name
- Set of built-in predicates
  - only one (boolean) solution (no open choices)
  - without non-determinism
- Set of built-in functors
- Theory
  - typically providing rules on top of the built-in predicates/functors
  - recovering non-determinism

Developing a Library

- Extending `alice.tuprolog.Library` abstract class

  provides some basic services

  - Unification: `boolean unify(Term arg0, Term arg1)`
- Writing built-in predicates as methods

  `boolean` *`predicateName_arityN`*`(Term arg1,..., Term argN)`

  - *fails* if exceptions are thrown

  more specific terms are allowed (see example)
- Writing built-in funtors as methods

  `Term` *`functorName_arityN`*`(Term arg1,..., Term argN)`

  - *fails* if exceptions are thrown
- Defining the theory

  overriding `String getTheory()` method

  default = empty theory = empty string
- Defining the name

  overriding `String getName()` method

  default = class name

Example

```java
package alice.test.tuprolog;

import alice.tuprolog.*;

public class StringLibrary extends Library {

  public boolean to_lower_case_2(Term arg0, Term arg1){
    String dest = arg0.toString().toLowerCase();
    return unify(arg1, new Struct(dest));
  }
}
```

Example (2)

```
package alice.test.tuprolog;

import alice.tuprolog.*;
import alice.tuprolog.Number;

public class TestLibrary extends StringLibrary {

  public Term sum_2(Number arg0, Number arg1){
    int sum = arg0.intValue()+arg1.intValue();
    return new Int(sum);
  }

  public String getTheory(){
    return “print_sum(A,B):- N is sum(A,B), \n”+
                 “write(N),nl. \n”;
}
```

Remarks

- Libraries as bridge to legacy libreries & code

  not only Java

  - exploiting JNI

- Solution for specialised & performant computation

  Math libraries

  Graphics 3D/2D libraries (e.g. OpenGL)

  Multimedia libraries

  - sound, video streaming, etc

Standard Libraries

- Package `alice.tuprolog.lib`
- Basic Library

  `alice.tuprolog.lib.BasicLibrary`

  basic & frequent builtins
- ISO Library

  `alice.tuprolog.lib.ISOLibrary`

  ISO standard builtins
- IO Library

  `alice.tuprolog.lib.IOLibrary`

  Input/Output Builtins (File, Console, etc)
- **Java Library**

  `alice.tuprolog.lib.JavaLibrary`

  Using Java from 2P

The JavaLibrary

- Goals

  reusing all the Java world *from Prolog*

  OOP-oriented access style

  - object creation
  - method invocation

  full dynamism

  - classes/objects created/linked at run-time

  Java Event Management (1.2.1 version)

JavaLibrary Design Goal

- Mapping 1-1 with Java operation

  simple object creation

  method invocation

  field access

- Keeping the two paradigms separated

  Object references are denotated by simple terms

  The terms refer to Java objects *only* in the context of JavaLibrary builtins predicates/operators

  - not embedding an OOP model inside the Prolog virtual machine

Simple class used for the examples

```
public class Counter {
  private int value;
  public  String name;

  public Counter(){}
  public Counter(String n){ name=n; }

  public void setValue(int val){ value=val; }
  public int  getValue() { return value; }
  public void inc()      { value++; }
  public void setName(String s) { name = s;}

  static public String getVersion() { return "1.0"; }
}
```

Creating Java Objects

- Syntax

  `java_object(`*`@ClassName,@ArgumentList,?ObjectRef`*`)`

- Semantics

  the predicate is true if *`ObjectRef`* denotates a new instance of a Java object of class *`ClassName`* created with arguments *`ArgumentList`*

- Example

```
?- java_object('Counter',[],MyCounter).
?- java_object('Counter',[],aCounter).
?- java_object('Counter',['pippo'],pippoCounter).
```

- For arrays

  `java_object(`*`@ClassName[],@ArgumentList,?ObjectRef`*`)`

  Example

```
?- java_object('Counter[]',[10],MyArray).
```

Method Invocation

- Syntax

```
@TargetRef <- @MethodName(@Arguments)
@TargetRef <- @MethodName(@Arguments) returns ?Result
class(@ClassName) <- @MethodName(...) ...
```

- Semantics

  the predicate is true if the method `MethodName` can be invoked on the object referenced by `TargetRef` possibly obtaining a result unifying with `Result`.

- Examples

```
?- …, MyCounter <- setValue(303).
?- …, pippoCounter <- getValue returns Value.
?- class('Counter') <- getVersion returns Version.
```

Accessing Fields

- Syntax

```
@TargetRef . @Field <- set(@Content)
@TargetRef . @Field <- get(?Content)
```

- Semantics

  the predicates are true if the content of the field *`Field`* of the object referenced by *`TargetRef`* can be set with the value *`Content`* or can be read and its value unifies the *`Content`* value.

- Examples

```
?- …, MyCounter.name <- get(Name).
?- …, pippoCounter.name <- set('pippo2').
```

Dynamic Compilation

- Compiling dynamically a Java class given its source
- Syntax
  `java_class(@Source,@FullClassName,@ClassPathList,-ClassRef)`
- Semantics

  the predicate is true if *ClassRef* is the reference to an object class whose name is *FullClassName* and it can be obtained by compiling the source *Source* with class pathes of the list *ClassPathList*.
- Example

JavaLibrary: Overall Example

```
test:-
    java_object('Counter',['ANiceCounter'],myCounter),
    myCounter <- setValue(5),
    myCounter <- inc,
    myCounter <- getValue returns Value,
    write(Value),
    class('Counter') <- getVersion returns Version,
    class('java.lang.System').out <- get(StdOut),
    StdOut <- println(Version),
    myCounter.name <- get(MyName),
    StdOut <- println(MyName),
    myCounter.name <- set('NicerCounter'),
    java_object('Counter[]', [10], ArrayCounters),
    java_array_set(ArrayCounters, 3, myCounter).
```

JavaLibrary: Examples (1)

Opening a swing dialog for choosing a file:

Java Library: Examples (2)

- Using RMI Infrastructure

```
?- java_object('java.rmi.RMISecurityManager',[],Manager),
   class('java.lang.System') <- setSecurityManager(Manager),
   class('java.rmi.Naming') <- lookup('pippoCounter') returns MyCounter,
   MyCounter <- inc,
   MyCounter <- getValue returns Val,
   write(Val),nl.
```

- In Java...

```
...
RMISecurityManager rman = new RMISecurityManager();
System.setSecurityManager(rman);
Counter core=(Counter)Naming.lookup("pippoCounter");
counter.inc();
long val = counter.getValue();
System.out.println(val);
```

JavaLibrary: Examples (3)

"Find the minimum path between two cities.
City distances are stored in a standard DB
(table 'distances', city_from/city_to/length row)"

```
find_path( From, To ):-
  init_dbase( 'jdbc:odbc:distances', Connection,'',''),
  exec_query( Connection,
     'SELECT city_from, city_to, distance FROM distances.txt',ResultSet ),
  assert_result( ResultSet ),
  findall( pa(Length,L), paths(From,To,L,Length), PathList ),
  current_prolog_flag( max_integer, Max ),
  min_path( PathList, pa(Max,_), pa(MinLength, MinList) ),
  outputResult( From, To, MinList, MinLength ).
```

JavaLibrary: Examples (3)

```
% Access to Database
init_dbase( DBase, Username, Password, Connection ) :-
    class('java.lang.Class') <- forName( 'sun.jdbc.odbc.JdbcOdbcDriver' ),
    class('java.sql.DriverManager') <- getConnection( DBase, Username, Password)
                                       returns Connection,
    write( '[ Database ' ), write(DBase), write(' connected ]' ), nl.

exec_query( Connection, Query, ResultSet ):-
    Connection <- createStatement returns Statement,
    Statement <- executeQuery( Query ) returns ResultSet,
    write( '[ query ' ), write(Query), write( ' executed ]' ), nl.

assert_result( ResultSet ) :-
    ResultSet <- next returns Valid, Valid == true, !,
    ResultSet <- getString( 'city_from' ) returns From,
    ResultSet <- getString( 'city_to' ) returns To,
    ResultSet <- getInt( 'distance' ) returns Dist,
    assert( distance(From, To, Dist) ),
    assert_result( ResultSet ).
    assert_result(_).
```

JavaLibrary: Examples (3)

```
paths( A, B, List, Length ):- path( A, B, List, Length, [] ).
path( A, A, [], 0, _).
path( A, B, [City|Cities], Length, VisitedCities):-
   distance( A, City, Length1 ),
   not( member(City, VisitedCities) ),
   path( City, B, Cities, Length2, [City|VisitedCities] ),
   Length is Length1 + Length2.

min_path( [], X, X) :- !.
min_path( [ pa(Length, List) | L ], pa(MinLen, MinList), Res):-
   Length < MinLen, !,
   min_path( L, pa(Length,List), Res ).

min_path( [_|MorePaths], CurrentMinPath, Res) :-
min_path( MorePaths, CurrentMinPath, Res).

writeList( [] ) :- !.
writeList( [X|L] ) :- write(','), write(X), !, writeList( L ).
```

JavaLibrary: Examples (3)

```
outputResult( From, To, [], _) :- !,
   write('no path found from '), write(From),
   write(' to '), write(To), nl.

outputResult( From, To, MinList, MinLength) :-
   write('min path from '), write(From),
   write(' to '), write(To), write(': '),
   write(From), writeList(MinList),
   write(' - length: '), write(MinLength).
```

Other JavaLibrary features

- Casting Reference Types
- Built-ins to access arrays
- Predefined Object Bindings

(Refer to 2P user manual for detailed information)

Thesis on 2P – Done

- 2Pme
  - 2P on Cellular phones (J2ME)
- 2Pstorm
  - 2P controlling LEGO Mindstorm Robots
- 2P on a web server?
- 2P site
  - Project

Thesis on 2P – Available

- Applications on top of 2Pme
- Extending 2P toward Exceptions
- Extending the JavaLibrary toward Java Events
- Improving 2P performance
- MAS on 2P
- Constraint mechanisms (ongoing)
- Modules / Contexts in 2P
- 3APL on 2P
- DALI on 2P