2P-Kt: logic programming with objects & functions in Kotlin

Giovanni Ciatto* Roberta Calegari° Enrico Siboni† Enrico Denti‡ Andrea Omicini⋆

*‡⋆Dipartimento di Informatica – Scienza e Ingegneria (DISI)
°Alma Mater Research Institute for Human-Centered Artificial Intelligence
Alma Mater Studiorum—Università di Bologna, Italy
{giovanni.ciatto, roberta.calegari, enrico.denti, andrea.omicini}@unibo.it

†University of Applied Sciences and Arts of Western Switzerland (HES-SO)
enrico.siboni@hevs.ch

21st Workshop "From Objects to Agents" (WOA)
Sept. 16, 2020, Bologna, Italy

Next in Line...

Context

AI side

- AI is shining, brighter than ever
  - mostly thanks to the advances in ML and sub-symbolic AI

⇒ symbolic AI is gaining momentum because of XAI

  ! hybrid solution mixing logic & data-driven AI are flourishing [3]

MAS side

The MAS community is eager for logic-based technologies [2]

- to support agents' knowledge representation, reasoning, or execution
- or to prove MAS properties

! despite few mature tech exist, and even fewer are actively maintained

Motivation

The problem with logic-based technologies

There is technological barrier slowing

- the adoption of logic programming (LP) as paradigm
- the exploitation of logic-based technologies

while programming *in the large*

- $\overbrace{\text{mainstream programming languages}}^{\textit{e.g. Scala, Kotlin, Python, C\#}}$ are blending several paradigms

  e.g. imperative, object-oriented (OOP), and functional programming (FP)

  - except LP!

- $\underbrace{\text{mainstream platforms}}_{\textit{e.g. JVM, .NET, JS, Python}}$ are poorly interoperable with logic-based tech.

Motivating example – SWI-Prolog's FLI for Java

- Prolog [4] implementors rely on Foreign Language Interfaces (FLI) [1]
  - (mostly targetting Java, or C)
- For instance, SWI-Prolog comes with a FLI for Java[1]:

```
Query query = new Query("parent", new Term[] {
      new Atom("adam"),
      new Variable("X")
    }
  ); // ?- parent(adam, X).
Map<String,Term> solution = query.oneSolution();
System.out.println("The child of Adam is " + solution.get("X"));
```

→ No paradigm harmonization between Prolog and $\underbrace{\text{the hosting language}}_{\textit{i.e. Java}}$

[1] https://jpl7.org

Contribution of the paper

- Show that OOP, FP, and LP can be blended into a single language
- Propose a DSL blending Kotlin (OOP + FP) and Prolog (LP)
- Pave the way to the creation of similar DSL in mainstream languages

Next in Line. . .

Focus on. . .

Whet your appetite

Our Kotlin DSL for Prolog vs. actual Prolog

```
prolog {
  staticKb(
    rule {
      "ancestor"(X, Y) `if` "parent"(X, Y)         // ancestor(X, Y) :- parent(X, Y).
    },
    rule {
      "ancestor"(X, Y) `if` (                      // ancestor(X, Y) :-
        "parent"(X, Z) and "ancestor"(Z, Y)        //     parent(X, Z), ancestor(Z, Y).
      )
    },
    fact { "parent"("abraham", "isaac") },         // parent(abraham, isaac).
    fact { "parent"("isaac", "jacob") },           // parent(isaac, jacob).
    fact { "parent"("jacob", "joseph") }           // parent(jacob, joseph).
  )

  for (sol in solve("ancestor"("abraham", D)))     // ?- ancestor(abraham, D),
    if (sol is Solution.Yes)                       //        write(D), nl.
      println(sol.substitution[D])
}
```

! try it here: https://github.com/tuProlog/prolog-dsl-example

Focus on. . .

Design Principles

P_1 – The DSL strictly extends the hosting language

→ no feature of the hosting language is forbidden within the DSL

P_2 – The DSL is interoperable with hosting language

→ all features of the hosting language are allowed within the DSL

→ LP is *harmonised* with the hosting language paradigm(s)

P_3 – The DSL is well encapsulated within the hosting language

→ i.e. only usable within well-identifiable sections

P_4 – The DSL is as close as possible to Prolog

→ both syntactically & semantically

Focus on. . .

Functioning in the Kotlin case I

The DSL is only enabled within `prolog {` $\langle$*DSL block*$\rangle$ `}` expressions

Expressions of the form: `"functor"(`$\langle e_1\rangle$`, `$\langle e_2\rangle$`, ...)`
are interpreted as terms: `functor(`t_1`, `t_2`, ...)`
provided that $\forall i : \langle e_i\rangle$ can be converted into t_i

Expressions of the form:
`rule {"head"(`$\langle e_1\rangle$`, ..., `$\langle e_N\rangle$`) `if` (`$\langle e_{N+1}\rangle$` and ... and `$\langle e_M\rangle$`) }`
are interpreted as rules: `head(`t_1`, ..., `t_N`) :- `t_{N+1}`, ..., `t_M
provided that $\forall i : \langle e_i\rangle$ can be converted into t_i

- similar syntax for facts

Functioning in the Kotlin case II

Within `prolog { ... }` blocks

- `staticKb(`*Clause*$_1$`, `*Clause*$_2$`, ...)` sets up the local static KB
- `dynamicKb(`*Clause*$_1$`, `*Clause*$_2$`, ...)` sets up the local dynamic KB
- `solve(`*Query*`, `*Timeout*`)` returns a *lazy stream* of solutions
- `assert(`*Clause*`)` appends a new clause to the local dynamic KB
- ...

Kotlin to Prolog conversions

Kotlin	Prolog
lowercase string	atom
uppercase string	variable
int, long, short, byte	integer
double, float	real
boolean	atom
list, array, iterable	list

Next in Line. . .

Recipe for a Prolog-like DSL

1. A language with a flexible API
2. Full fledged API for Prolog, supporting that language
3. Exploit flexibility to hide the exploitation of the API

Kotlin mechanisms

1. Operator overloading
2. Block-like lambda expressions
3. Function types/literals with receivers
4. Extension methods

2P-Kt – Overview

1. Operator overloading
2. Block-like lambda expressions
3. Function types/literals with receivers
4. Extension methods

DSL design on top of 2P-Kt

1. Onion scopes
2. Layered views

Next in Line. . .

Conclusions & future works

Summing up

Summarise the most relevant contributions of this study:

- conclusion 1
- conclusion 2
- conclusion 3

Future works

Sketch some future research directions

- future work 1
- future work 2

(may be split into 2 slides)

2P-Kt: logic programming with objects & functions in Kotlin

Giovanni Ciatto* Roberta Calegari° Enrico Siboni†
Enrico Denti‡ Andrea Omicini⋆

*‡⋆Dipartimento di Informatica – Scienza e Ingegneria (DISI)
°Alma Mater Research Institute for Human-Centered Artificial Intelligence
Alma Mater Studiorum—Università di Bologna, Italy
{giovanni.ciatto, roberta.calegari, enrico.denti, andrea.omicini}@unibo.it

†University of Applied Sciences and Arts of Western Switzerland (HES-SO)
enrico.siboni@hevs.ch

21st Workshop "From Objects to Agents" (WOA)
Sept. 16, 2020, Bologna, Italy

References I

[1] Roberto Bagnara and Manuel Carro.
Foreign language interfaces for Prolog: A terse survey.
ALP Newsletter, 15(2), May 2002.

[2] Roberta Calegari, Giovanni Ciatto, Viviana Mascardi, and Andrea Omicini.
Logic-based technologies for multi-agent systems: A systematic literature review.
Autonomous Agents and Multi-Agent Systems, In press.
Special Issue "Current Trends in Research on Software Agents and Agent-Based Software Development".

[3] Roberta Calegari, Giovanni Ciatto, and Andrea Omicini.
On the integration of symbolic and sub-symbolic techniques for XAI: A survey.
Intelligenza Artificiale, In press.

[4] Alain Colmerauer and Philippe Roussel.
The birth of prolog.
In John A. N. Lee and Jean E. Sammet, editors, *History of Programming Languages Conference (HOPL-II)*, pages 37–52. ACM, April 1993.