A second life for Prolog

Prolog as unifying framework

Jan Wielemaker
J.Wielemaker@cwi.nl

This research was partially supported by the VRE4EIC project, a project that has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 676247.
Overview

- YASL: Yet Another Scripting Language. Why?
- Interfaces to systems and languages
- Web services
- Exercises
YASL

- Scripting languages make low-level efficient algorithms available in a much more versatile manner than packed as command line tools. Notably, they allow for smaller granularity.

- Virtually all scripting languages combine imperative, functional and object-oriented aspects:
  - ✔ Versatile and familiar
  - ✗ Object-relational impedance mismatch
  - ✗ Single moded functions vs. multi-moded relations
  - ✗ No declarative reading
  - ✗ Generally more verbose
  - ✗ Weaker for defining DSLs (Domain Specific Languages)
Prolog as scripting language

✔ Relational data model is core
✔ Tree and graph models fit naturally
✔ Pattern matching (popular, see e.g., awk, sed, Perl)
✔ Sequence/gammars on arbitrary objects
✔ Excelent DSL definition capabilites
✔ Concise programs make it easy to graps semantics
✗ Poor handling of arrays
✗ Not widely known, very different from imperative languages
RDF (Linked Data, Knowledge Graphs)

- API
  - `rdf(?Subject, ?Predicate, ?Object, ?Graph)`
  - `{ Filters }`
- For example
  - `:- rdf_prefix(dbo, 'http://dbpedia.org/ontology/').`
  - `?- { Born > 2000 },
    rdf(Person, dbo:birthDate, Born).`
- Filters are constraints
  - Allows low-level database to exploit indexing
  - Allows combining multiple filters efficiently
RDF Backends

- Built-in library (semweb/rdf11)
  - C-based in-memory graph store
  - Aims at volatile RDF. Supports transactions and snapshots.
  - Limited to approx. 100,000,000 triples

- HDT (Header Dictionary Triples)
  - C++ library to compile and query static triple collections
  - Small memory footprint for very large graphs
RDF Misc

- ClioPatria web server
- Management and exploration web interface
- SPARQL server
- RDF parsers (RDF/XML, Turtle, ntriples, nquads, ...)
- SPARQL client
  - Native: `sparql_query(+Query, -Row, +Options)`
  - Sparkle:
    ```prolog
    :- dbp ?? rdf(Person,rdf:type,foaf:'Person'),
       rdf(Person,foaf:Name,Name),
       filter(regex('Colt.*',Name)).
    ```
Accessing databases

- ODBC
  - Native: odbc_query(DB, SQLQuery, Row)
  - CQL: high level specification that is compiled to SQL
- Key-Value stores
  - RocksDB
  - BerkeleyDB
Document processing

- SGML/XML/HTML5 parser
- Document based: element(Tag, Attributes, Content)
- Callback based: on_begin, on_cdata, etc.
- Mixed: trap on_begin for e.g., a record and read the record
- xpath/3 finds nodes in a document tree
R integration

• R provides access to a vast amount of statistical and machine learning algorithms and is capable of producing graphical output in many formats.

• Three interfaces
  
  • Rsession (Nicos Angelopoulos)
    • Uses piped to connect to R console. Mostly outdated.
  
  • Real (Nicos Angelopoulos, Vitor Santos Costa)
    • Uses Prolog and R C-interfaces. Great for local usage.
  
  • Rserve-client (Jan Wielemaker)
    • Uses binary Rserve client/server protocol. Great for (web) servers.

• https://swish.swi-prolog.org/example/Rdownload.swinb
R interface strategy

- Rsession: pure Prolog, tries to capture as much as possible of R.
- Real: SWI-Prolog 7 extensions, allowing for native support of a.b, f() and v[i]. Tries to capture as much as possible of R.
- Rserve-client: SWI-Prolog 7 extensions. Handles more advanced R through *Quasi Quotations*:

```
{|r(Param,...) || R code |}
```
Native (Web) services

- HTTP server library with **pluggable** components
  - OpenSSL (HTTPS)
  - Authentication (Basic, Digest, Oauth2, ...)
  - Sessions
  - Logging
  - Websockets
  - Static file service
  - Location dispatching (bind path to predicate)
  - JSON and XML read/write
  - HTML replies (generate HTML pages)
Native webservice

:- http_handler('/hello', hello, []).

hello(_Request) :-
  format('Content-type: text/plain~n~n'),
  format('Hello world!~n').

Link /hello to the predicate hello/1

Write GCI body
Native webservice (HTML)

:- http_handler('/hello', hello, []).

hello(_Request) :-
    reply_html_page(navigation,
        title('Hello world'),
        [ h1('Hello world'),
          say_hello
        ]).

say_hello -->
    html([ p(class(intro), 'This is my first paragraph') ]).
Native webservice (Ajax)

:- http_handler('/compute', compute, []).

compute(Request) :-
  http_read_json_dict(Request, JSONIn),
  compute(JSONIn, JSONOut),
  http_reply_json(JSONOut).
Pengines: Prolog Engines on the Web

- Access Prolog *toplevel* using HTTP and JSON (or Prolog)
  - Create (from source)
    - Returns ID
  - Ask (query, N)
    - Returns error or first N answers
  - Next (N)/Stop/Abort
    - Returns error or next N answers
  - Destroy
    - Dispose of the engine
Penguins

- Bring **program to the data**. Configuration chooses between authenticated or anonymous access. Anonymous access is always sandboxed.
- Universal API to any service.
- Clients:
  - Prolog
  - JavaScript (nodejs)
  - Java
  - Erlang
  - Ruby
  - Shell (curl, returning all answers)
Pengines and SWISH

- SWISH is a JavaScript application using Pengines
- Server provides storage, highlighting support, help, etc.
- Allows shared web-based development of programs specific to data stored on the SWISH instance
- Pengine clients can access the stored programs
Wrap up
What went wrong with Prolog 1.0

- Politics (Japan/US)?
- Too *alien* for people with CS background?
  - No functions, no „normal“ variables, no arrays, no loops, …
- Not scalable / toy language / only for puzzles?
- Fragmentation / poor standardization?
- Relational impedance also applies for embedding Prolog
# Prolog niches (speculative)

<table>
<thead>
<tr>
<th>Prolog</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWI-Prolog</td>
<td>Networking, system interfaces, (WEB) UI, IDE, concurrency, scalability, robustness</td>
</tr>
<tr>
<td>gprolog</td>
<td>clp(fd), native code embedding</td>
</tr>
<tr>
<td>YAP</td>
<td>Performance, integration with machine learning, scalability</td>
</tr>
<tr>
<td>XSB</td>
<td>SGL-WAM, deductive database, negation, well formed semantics</td>
</tr>
<tr>
<td>ECLiPSe</td>
<td>Constraint solving</td>
</tr>
<tr>
<td>SICStus</td>
<td>Constraints, performance, standard compliance, robustness</td>
</tr>
</tbody>
</table>
Some recent projects

- Business rule enforcements in finance
- Watson: interpret NLP parse trees
- Robotics: connect to large knowledge graphs
- Understanding changes in satellite images
- Program verification and test generation
- (Java) program analysis and refactoring
- Control parcel sorting equipment
- Natural language understanding
Future developments

• Technical
  • Improved indexing
  • High performance exchange and storage of terms
  • Improve SWISH
    • UI enhancements
    • Support for reproducible results (permalink)

• Long term
  • Scalable concurrent and distributed computing
  • Combine symbolic and statistical techniques
Prolog 2.0?

- Prolog is still as alien and the relational impedance mismatch that harms low-granularity embedding is not gone.
- The connection to statistical AI is unclear, but the need to unite symbolic and statistical AI is widely recognised.
- Prolog has evolved from pure SLD inference to a platform where SLD resolution is used to integrate more powerful inference mechanisms (declarative islands).
- (SWI-)Prolog has rich interfaces to languages, document formats and protocols.
- (SWI-)Prolog supports service based architectures well.
Take home

- Alain Colmerauer’s invention is very much alive
- Modern Prolog systems
  - Are scalable in terms of storage and concurrency
  - Extend SLD with (in part programmable) control
  - Have a rich set of interfaces to languages, network protocols and document formats