Towards an Implementation-Independent Interface for Reasoning about Semantic Web in Prolog

Hashimoto, Daniel Kiyoshi

3rd Scryer Prolog Meetup – November 2025
Hochschule Düsseldorf, Düsseldorf, Germany

Title Break Down

- Semantic Web

Title Break Down

- Semantic Web
- for Reasoning about … in Prolog

Title Break Down

- Semantic Web
- for Reasoning about … in Prolog
- an Implementation-Independent Interface

Title Break Down

- Semantic Web
- for Reasoning about … in Prolog
- an Implementation-Independent Interface
- Towards

Semantic Web

The Web of Documents

https://en.wikipedia.org/wiki/Prolog

https://en.wikipedia.org/wiki/Prolog

Prolog

59 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about the programming language. For the narrative device, see Prologue. For other uses, see Prologue (disambiguation).

Prolog is a logic programming language that has its origins in artificial intelligence, automated theorem proving, and computational linguistics.[1][2][3]

Prolog has its roots in first-order logic, a formal logic. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language: the program is a set of facts and rules, which define relations. A computation is initiated by running a *query* over the program.[4]

Prolog was one of the first logic programming languages[5] and remains the most popular such language today, with several free and commercial implementations available. The language has been used for theorem proving,[6] expert systems,[7] term rewriting,[8] type systems,[9], automated planning,[10], and question answering[11][12][13] as well as its original intended field of use, natural language processing.[14][11]

Prolog is a Turing-complete, general-purpose programming language, which is well-suited for intelligent knowledge-processing applications.

Prolog	
Paradigm	Logic
Designed by	Alain Colmerauer
First appeared	1972; 53 years ago
Stable release	Part 1: General core-Edition 1 (June 1995; 30 years ago) Part 2: Modules-Edition 1 (June 2000; 25 years ago) Part 3: Definite clause grammar rules (June 2025; 4 months ago)
Typing discipline	Untyped (its single data type is "term")
Filename extensions	`.pl`, `.pro`, `.P`
Website	Part 1: www.iso.org/

Prolog	
Paradigm	Logic
Designed by	Alain Colmerauer
First appeared	1972; 53 years ago
Stable release	Part 1: General core-Edition 1 (June 1995; 30 years ago) Part 2: Modules-Edition 1 (June 2000; 25 years ago) Part 3: Definite clause grammar rules (June 2025; 4 months ago)
Typing discipline	Untyped (its single data type is "term")
Filename extensions	`.pl`, `.pro`, `.P`
Website	Part 1: www.iso.org/standard/21413.html Part 2: www.iso.org/standard/20775.html Part 3: www.iso.org/standard/83635.html

Filename extensions	`.pl`, `.pro`, `.P`
Website	Part 1: www.iso.org/standard/21413.html Part 2: www.iso.org/standard/20775.html Part 3: www.iso.org/standard/83635.html

Major implementations
Amzi! Prolog, B-Prolog, Ciao, ECLiPSe, GNU Prolog, LPA Prolog, Poplog, P#, Quintus Prolog, Scryer Prolog, SICStus, Strawberry, SWI-Prolog, Tau Prolog, tuProlog, WIN-PROLOG XSB, YAP.
Dialects
ISO Prolog, Edinburgh Prolog
Influenced by
Planner
Influenced
CHR, Clojure, Datalog, Erlang, Epilog, KL0, KL1, Logtalk, Mercury, Oz, Strand, Visual Prolog

Prolog at Wikibooks

The data on the web should be machine-readable

A program should be able to easily:

- extract/read the data

The data on the web should be ~~machine-readable~~ machine-understandable

A program should be able to easily:

- extract/read the data
- reason about the data:
  - find more about the data
  - infer new data

The Web of ~~Documents~~ Data

The data on the web should be ~~machine-readable~~ machine-understandable

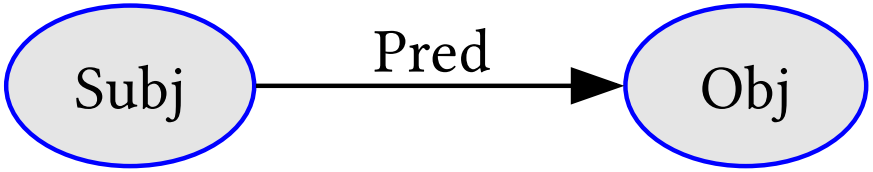
A program should be able to easily:

- extract/read the data
- reason about the data:
  - find more about the data
  - infer new data

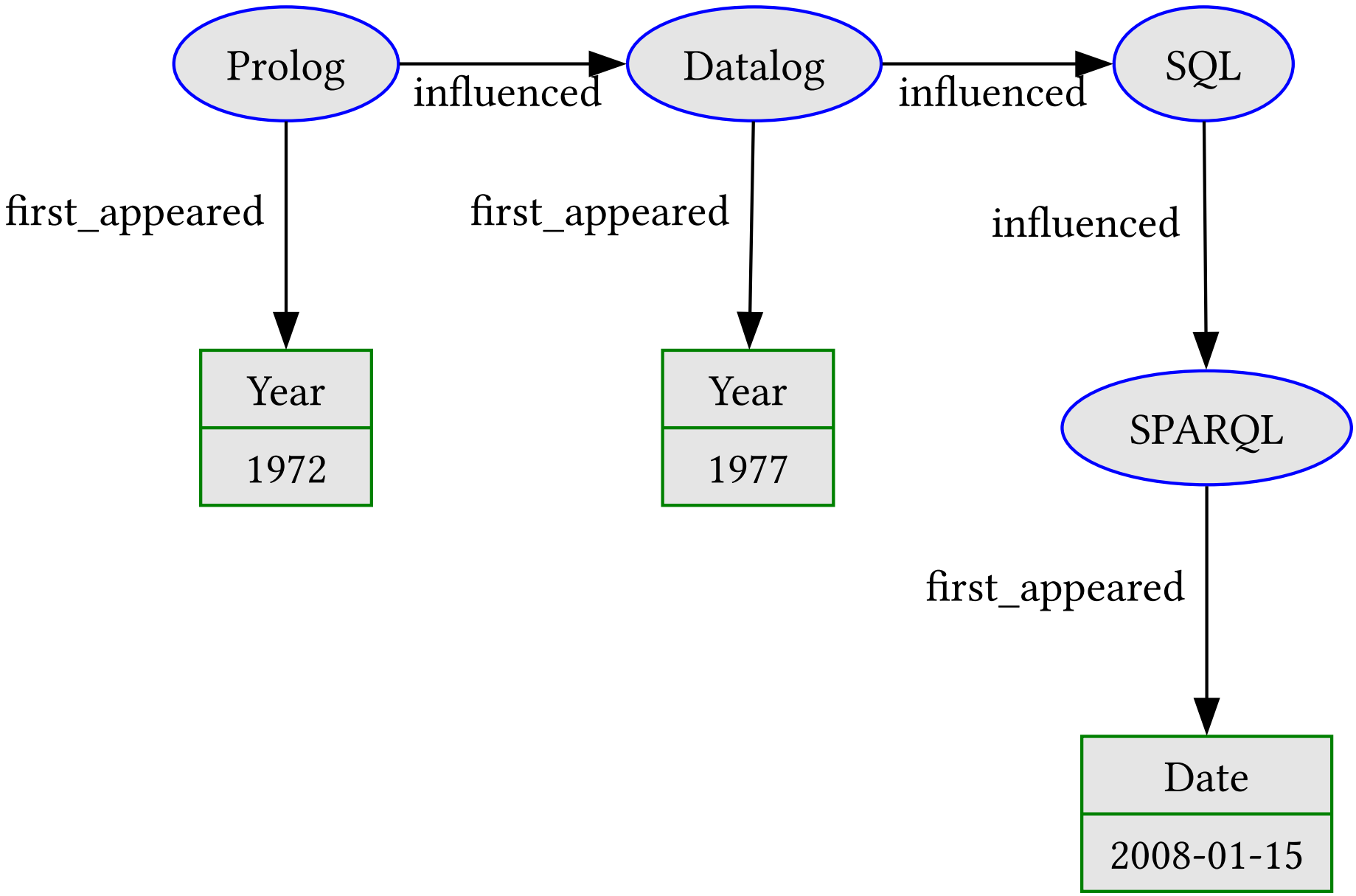
We need to add semantic information to the data on the web!

Knowledge representation is a directed graph with edge labels:

- nodes are resources
- directed edges are predicates
- the Predicate holds for Subject and Object:



A RDF Graph is a set of triples.


Prolog
influenced
Datalog
influenced
SQL
first_appeared
first_appeared
influenced
Year
1972
Year
1977
SPARQL
first_appeared
Date
2008-01-15

RDF Serialization Formats

There are many serialization options for RDF (most are in plain text):

- RDF/XML [text]
- **Turtle** [text/human-readable]
- HTML+RDFa (RDF embedding into HTML) [text]
- HDT (Header-Dictionary-Triples) [binary]
- ...

There are many serialization options for RDF (most are in plain text):

- RDF/XML [text]
- **Turtle** [text/human-readable]
- HTML+RDFa (RDF embedding into HTML) [text]
- HDT (Header-Dictionary-Triples) [binary]
- ...

```
@prefix : <http://www.example.org/> .
@prefix wiki: <http://www.wikipedia.org/wiki/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

wiki:Prolog
  :first_appeared "1972"^^xsd:gYear ;
  :influenced <http://www.wikipedia.org/wiki/Datalog> .
wiki:Datalog
  :first_appeared "1977"^^xsd:gYear ;
  :influenced wiki:SQL .
wiki:SQL :influenced wiki:SPARQL .
wiki:SPARQL :first_appeared "2008-01-15"^^xsd:date .
```

SPARQL is “SQL for RDF Graphs”

Pattern-matching on triples

Queries

SPARQL is “SQL for RDF Graphs”
Pattern-matching on triples

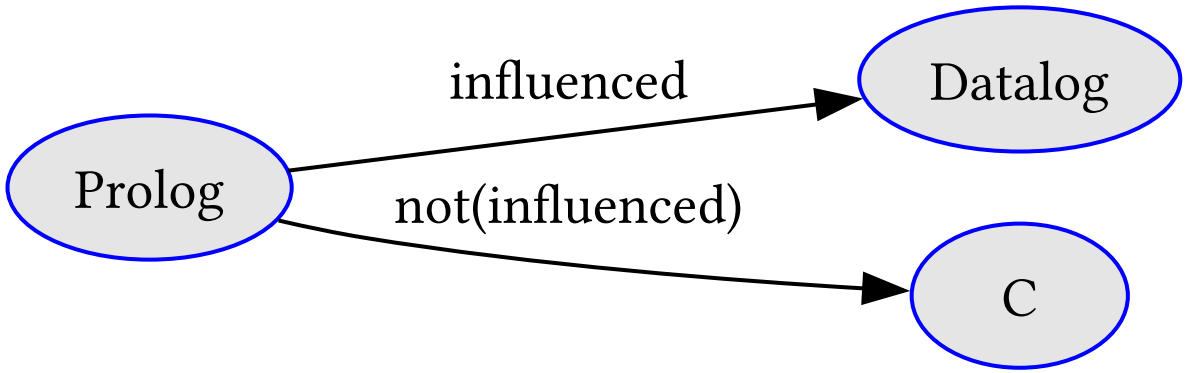
Caveats:

- We can query for Predicates

SPARQL is “SQL for RDF Graphs”
Pattern-matching on triples

Caveats:

- We can query for Predicates
- **Open World Assumption** vs. Closed World Assumption
  - negative information is another triple:



Machines use ontologies to make inferences

OWL2 (Web Ontology Language) is based on Description Logics

Machines use ontologies to make inferences

OWL2 (Web Ontology Language) is based on Description Logics

Ontology is a describes Predicates

Example of rules:

- A Predicate is reflexive, symmetric, transitive, ...
- A Predicate's domain and range

Interface

Existing Libraries

ClioPatria (SWI-Prolog) [cliopatria.swi-prolog.org/home]

- focus on RDF/SPARQL queries
- the triple-store is a c-extension

Thea2 (SWI-Prolog/ISO-Prolog) [vangelisv.github.io/thea]

- focus on graphs with ontologies
- a collection of file-libraries

My Proof of Concept Library [gitlab.com/Hashi364/semweb]

- focus on accurate representation of RDF concepts

Use Cases

Use Cases:

- Graph Analysis (Query)
- Graph Construction/Modification (Inference)

Use Cases

Use Cases:

- Graph Analysis (Query)
- Graph Construction/Modification (Inference)

We need to choose good representations and interfaces for:

- Resources
- Knowledge Graphs
- Queries

Resources

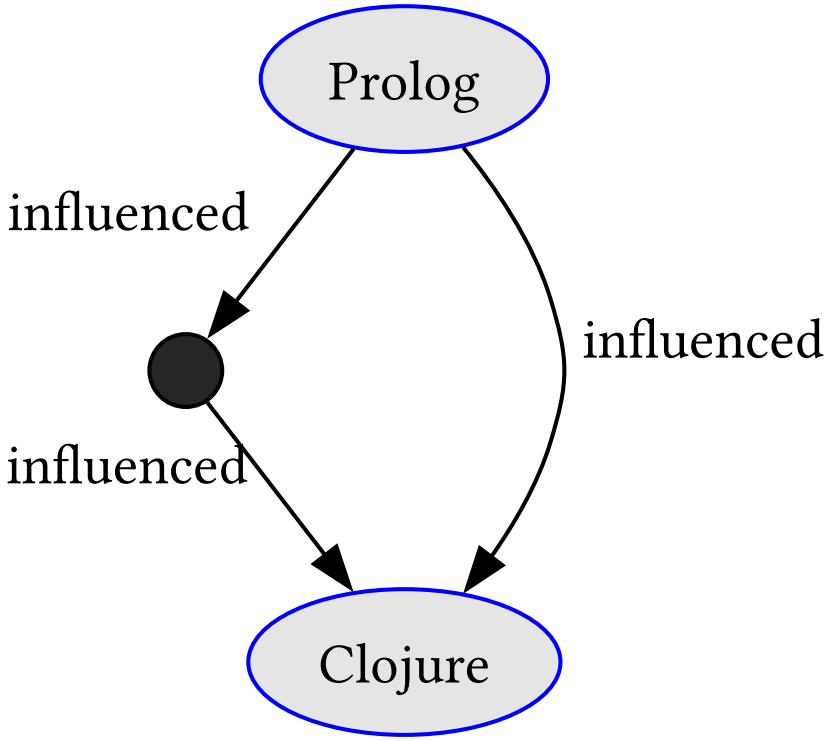
There are 3 kinds of resources:

- IRIs (links)
- Literals (strings, numbers, dates, …)
- Blank Nodes
  - dummy nodes
  - “existentialy quantified” nodes

Resources

There are 3 kinds of resources:

- IRIs (links)
- Literals (strings, numbers, dates, …)
- Blank Nodes
  - dummy nodes
  - “existentialy quantified” nodes



IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs
any ground term	`lang(prolog)`	more flexibility	unification (deep terms)

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs
any ground term	`lang(prolog)`	more flexibility	unification (deep terms)
`iri(IRI)`	`iri('https://www.wikipedia.org/wiki/Prolog')`	pattern matching (IRI checking)	longer terms (hard to read)

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs
any ground term	`lang(prolog)`	more flexibility	unification (deep terms)
`iri(IRI)`	`iri('https://www.wikipedia.org/wiki/Prolog')`	pattern matching (IRI checking)	longer terms (hard to read)
`Ns:Frag` and `:(Frag)`	`wiki:prolog`, `:(prolog)`	readability	Namespaces and IRI collision

Blank node representation

Representation	Examples	Comment
atoms starting with `'_:'`	`'_:a'`	unification

Blank node representation

Representation	Examples	Comment
atoms starting with `'_:'`	`'_:a'`	unification
no support (use IRIs)		IRI generation (skolemization)

Blank node representation

Representation	Examples	Comment
atoms starting with `'_:'`	`'_:a'`	unification
no support (use IRIs)		IRI generation (skolemization)
`blank(Labeled, Name)`	`blank(labeled, '_:a')`, `blank(unlabeled, foo(bar, baz))`	more flexible, pattern matching

Literal representation

Representation	Examples	Comment
literal(Lit)	`literal(1)`, `literal(@("prolog", "en"))`	close to prolog semantics

Literal representation

Representation	Examples	Comment
literal(Lit)	`literal(1),` `literal(@("prolog", "en"))`	close to prolog semantics

literal(Type, Repr)
^^(Repr, Type)
	`literal(IntegerIRI, "1"),`
	`literal(IntegerIRI, "01"),`
	`literal(LangStrIRI, @("prolog", "en"))`
		closer to RDF semantics

Using IRI atoms representation:

```
1 IntegerIRI = 'http://www.w3.org/2001/XMLSchema#integer',
2 LangStrIRI = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#langString'.
```

A RDF Graph is a set of triples

Interface	Examples
assert/retract	`assert_axiom(prolog, influenced, datalog).`
(implicit graph)	`retract_axiom(prolog, influenced, datalog).`

A RDF Graph is a set of triples

Interface	Examples
assert/retract (implicit graph)	`assert_axiom(prolog, influenced, datalog).` `retract_axiom(prolog, influenced, datalog).`
custom datatype (explicit graph)	`put_axiom(prolog, influenced, datalog, G0, G).` `del_axiom(prolog, influenced, datalog, G0, G).`

Explicit graph allows:

- working with multiple graphs
- set operations (union, intersection, minus, ...)

Interface	Example
inline queries	`?- Pred = influenced, rdf(prolog, Pred, X), rdf(X, Pred, Y).`

Queries

Interface	Example
inline queries	`?- Pred = influenced, rdf(prolog, Pred, X), rdf(X, Pred, Y).`
Domain Specific Language (similar to DCGs)	`?- Pred = influenced, query(( rdf(prolog, Pred, X), rdf(X, Pred, Y) )).`

DSL helps to achieve:

- load-time optimizations (`term_expansion/2` and `goal_expansion/2`)
- SPARQL translation (to and from)
  - federation queries

About My Library and Me

About My Web Semantic Experience

Semantic Web Course (2025 March $\sim$ July)

- Late-undergraduate/Graduate level
- I wrote the Library in less than 3 weeks (final assignment)
  - unordered list of triples for the triple store
  - most of the implementation relies on `library(lists)` and `library(reif)`
- As far as I know, there are 0 users

About My Web Semantic Experience

Semantic Web Course (2025 March ∼ July)

- Late-undergraduate/Graduate level
- I wrote the Library in less than 3 weeks (final assignment)
  - unordered list of triples for the triple store
  - most of the implementation relies on `library(lists)` and `library(reif)`
- As far as I know, there are 0 users

I have a static Turtle generator (unrelated with my library)

- `github.com/Kiyoshi364/static-memory` (my personal "web site")

About My Web Semantic Experience

Semantic Web Course (2025 March ∼ July)

- Late-undergraduate/Graduate level
- I wrote the Library in less than 3 weeks (final assignment)
  - unordered list of triples for the triple store
  - most of the implementation relies on `library(lists)` and `library(reif)`
- As far as I know, there are 0 users

I have a static Turtle generator (unrelated with my library)

- `github.com/Kiyoshi364/static-memory` (my personal "web site")

Reference links for Semantic Web:

- `github.com/semantalytics/awesome-semantic-web`

About Me

Graduate Student at UFRJ (Brazil)

- My research field is in Programming Languages
- Formalizing connections between Applicative and Concatenative Tacit Programming
  - Combinatory Logic and Concatenative Calculus
  - Publications: `github.com/Kiyoshi364/static-memory`

About Me

Graduate Student at UFRJ (Brazil)

- My research field is in Programming Languages
- Formalizing connections between Applicative and Concatenative Tacit Programming
  - Combinatory Logic and Concatenative Calculus
  - Publications: `github.com/Kiyoshi364/static-memory`

Research interests:

- alternative programming paradigms and computing models
- theorem proving and proof assistants
- static analysis/type systems/logic systems
- creating and using models
- “point at two things and saying ‘they are equal!’”

Towards an Implementation-Independent Interface for Reasoning about Semantic Web in Prolog

Hashimoto, Daniel Kiyoshi

3rd Scryer Prolog Meetup – November 2025
Hochschule Düsseldorf, Düsseldorf, Germany

Title Break Down

- Semantic Web

Title Break Down

- Semantic Web
- for Reasoning about … in Prolog

Title Break Down

- Semantic Web
- for Reasoning about … in Prolog
- an Implementation-Independent Interface

Title Break Down

- Semantic Web
- for Reasoning about … in Prolog
- an Implementation-Independent Interface
- Towards

Semantic Web

The Web of Documents

https://en.wikipedia.org/wiki/Prolog

https://en.wikipedia.org/wiki/Prolog

Prolog

59 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about the programming language. For the narrative device, see Prologue. For other uses, see Prologue (disambiguation).

Prolog is a logic programming language that has its origins in artificial intelligence, automated theorem proving, and computational linguistics.[1][2][3]

Prolog has its roots in first-order logic, a formal logic. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language: the program is a set of facts and rules, which define relations. A computation is initiated by running a *query* over the program.[4]

Prolog was one of the first logic programming languages[5] and remains the most popular such language today, with several free and commercial implementations available. The language has been used for theorem proving,[6] expert systems,[7] term rewriting,[8] type systems,[9], automated planning,[10], and question answering[11][12][13] as well as its original intended field of use, natural language processing.[14][11]

Prolog is a Turing-complete, general-purpose programming language, which is well-suited for intelligent knowledge-processing applications.

Prolog	
Paradigm	Logic
Designed by	Alain Colmerauer
First appeared	1972; 53 years ago
Stable release	Part 1: General core-Edition 1 (June 1995; 30 years ago) Part 2: Modules-Edition 1 (June 2000; 25 years ago) Part 3: Definite clause grammar rules (June 2025; 4 months ago)
Typing discipline	Untyped (its single data type is "term")
Filename extensions	`.pl`, `.pro`, `.P`
Website	Part 1: www.iso.org/

Prolog	
Paradigm	Logic
Designed by	Alain Colmerauer
First appeared	1972; 53 years ago
Stable release	Part 1: General core-Edition 1 (June 1995; 30 years ago) Part 2: Modules-Edition 1 (June 2000; 25 years ago) Part 3: Definite clause grammar rules (June 2025; 4 months ago)
Typing discipline	Untyped (its single data type is "term")
Filename extensions	`.pl`, `.pro`, `.P`
Website	Part 1: www.iso.org/standard/21413.html Part 2: www.iso.org/standard/20775.html Part 3: www.iso.org/standard/83635.html

Filename extensions	`.pl`, `.pro`, `.P`
Website	Part 1: www.iso.org/standard/21413.html Part 2: www.iso.org/standard/20775.html Part 3: www.iso.org/standard/83635.html

Major implementations
Amzi! Prolog, B-Prolog, Ciao, ECLiPSe, GNU Prolog, LPA Prolog, Poplog, P#, Quintus Prolog, Scryer Prolog, SICStus, Strawberry, SWI-Prolog, Tau Prolog, tuProlog, WIN-PROLOG XSB, YAP.
Dialects
ISO Prolog, Edinburgh Prolog
Influenced by
Planner
Influenced
CHR, Clojure, Datalog, Erlang, Epilog, KL0, KL1, Logtalk, Mercury, Oz, Strand, Visual Prolog

Prolog at Wikibooks

The data on the web should be machine-readable

A program should be able to easily:

- extract/read the data

The data on the web should be ~~machine-readable~~ machine-understandable

A program should be able to easily:

- extract/read the data
- reason about the data:
  - find more about the data
  - infer new data

The Web of ~~Documents~~ Data

The data on the web should be ~~machine-readable~~ machine-understandable

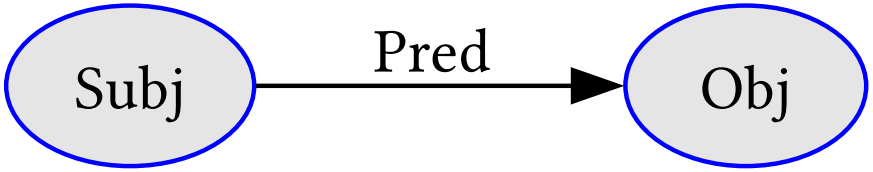
A program should be able to easily:

- extract/read the data
- reason about the data:
  - find more about the data
  - infer new data

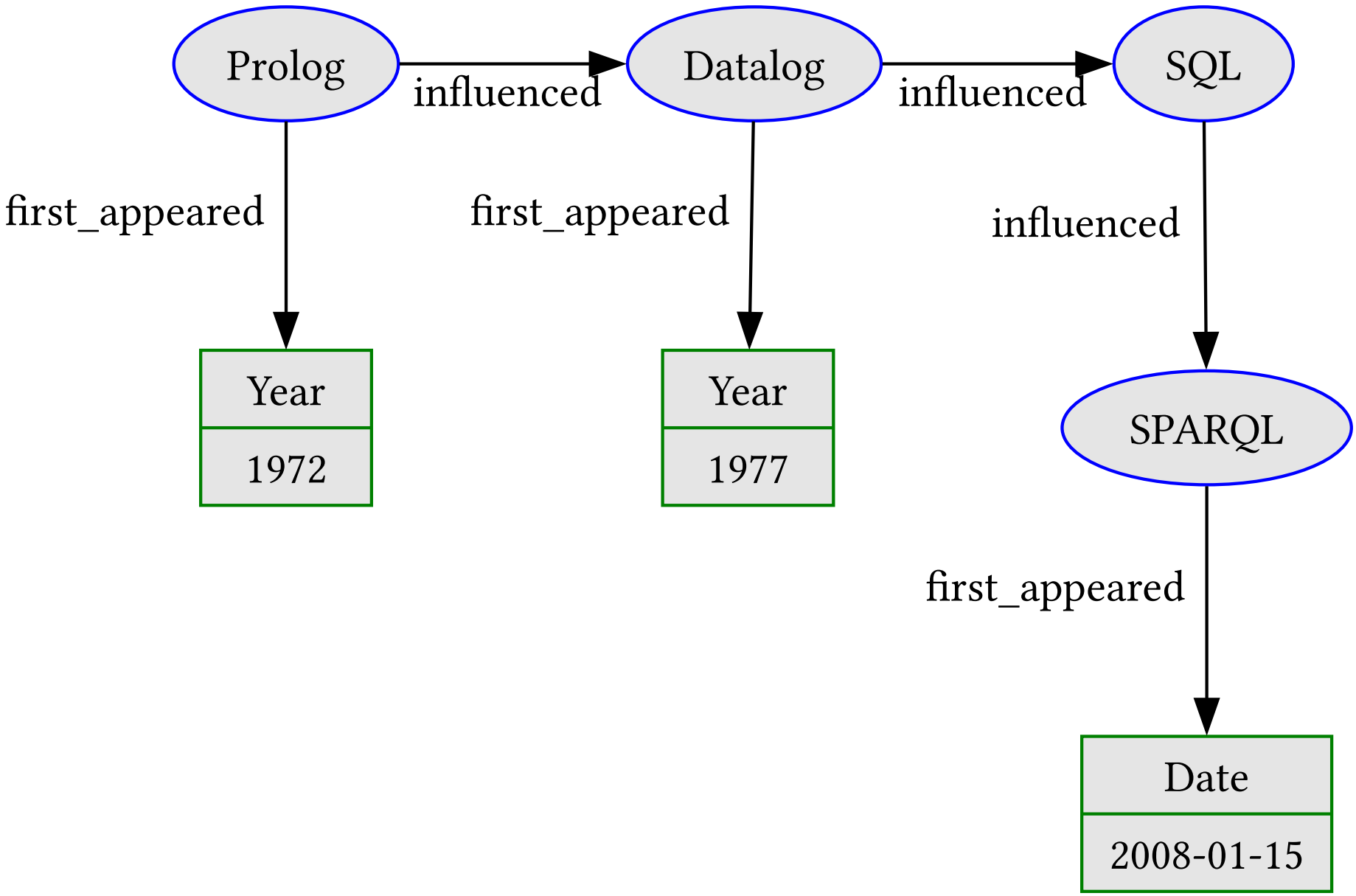
We need to add semantic information to the data on the web!

Knowledge representation is a directed graph with edge labels:

- nodes are resources
- directed edges are predicates
- the Predicate holds for Subject and Object:



A RDF Graph is a set of triples.


Prolog
influenced
Datalog
influenced
SQL
first_appeared
first_appeared
influenced
Year
1972
Year
1977
SPARQL
first_appeared
Date
2008-01-15

RDF Serialization Formats

There are many serialization options for RDF (most are in plain text):

- RDF/XML [text]
- **Turtle** [text/human-readable]
- HTML+RDFa (RDF embedding into HTML) [text]
- HDT (Header-Dictionary-Triples) [binary]
- ...

There are many serialization options for RDF (most are in plain text):

- RDF/XML [text]
- **Turtle** [text/human-readable]
- HTML+RDFa (RDF embedding into HTML) [text]
- HDT (Header-Dictionary-Triples) [binary]
- ...

```
@prefix : <http://www.example.org/> .
@prefix wiki: <http://www.wikipedia.org/wiki/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

wiki:Prolog
  :first_appeared "1972"^^xsd:gYear ;
  :influenced <http://www.wikipedia.org/wiki/Datalog> .
wiki:Datalog
  :first_appeared "1977"^^xsd:gYear ;
  :influenced wiki:SQL .
wiki:SQL :influenced wiki:SPARQL .
wiki:SPARQL :first_appeared "2008-01-15"^^xsd:date .
```

SPARQL is “SQL for RDF Graphs”

Pattern-matching on triples

Queries

SPARQL is “SQL for RDF Graphs”
Pattern-matching on triples

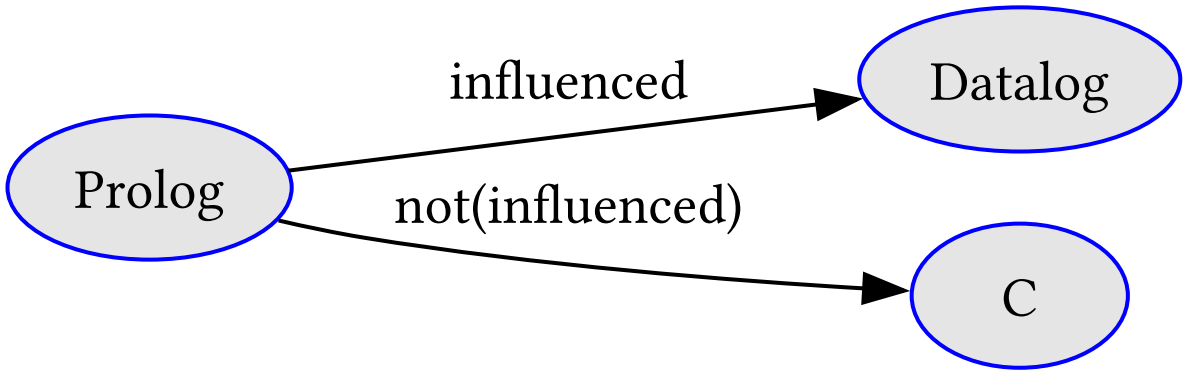
Caveats:

- We can query for Predicates

SPARQL is “SQL for RDF Graphs”
Pattern-matching on triples

Caveats:

- We can query for Predicates
- **Open World Assumption** vs. Closed World Assumption
  - negative information is another triple:



Machines use ontologies to make inferences

OWL2 (Web Ontology Language) is based on Description Logics

Machines use ontologies to make inferences

OWL2 (Web Ontology Language) is based on Description Logics

Ontology is a describes Predicates

Example of rules:

- A Predicate is reflexive, symmetric, transitive, ...
- A Predicate's domain and range

Interface

Existing Libraries

ClioPatria (SWI-Prolog) [cliopatria.swi-prolog.org/home]

- focus on RDF/SPARQL queries
- the triple-store is a c-extension

Thea2 (SWI-Prolog/ISO-Prolog) [vangelisv.github.io/thea]

- focus on graphs with ontologies
- a collection of file-libraries

My Proof of Concept Library [gitlab.com/Hashi364/semweb]

- focus on accurate representation of RDF concepts

Use Cases

Use Cases:

- Graph Analysis (Query)
- Graph Construction/Modification (Inference)

Use Cases

Use Cases:

- Graph Analysis (Query)
- Graph Construction/Modification (Inference)

We need to choose good representations and interfaces for:

- Resources
- Knowledge Graphs
- Queries

Resources

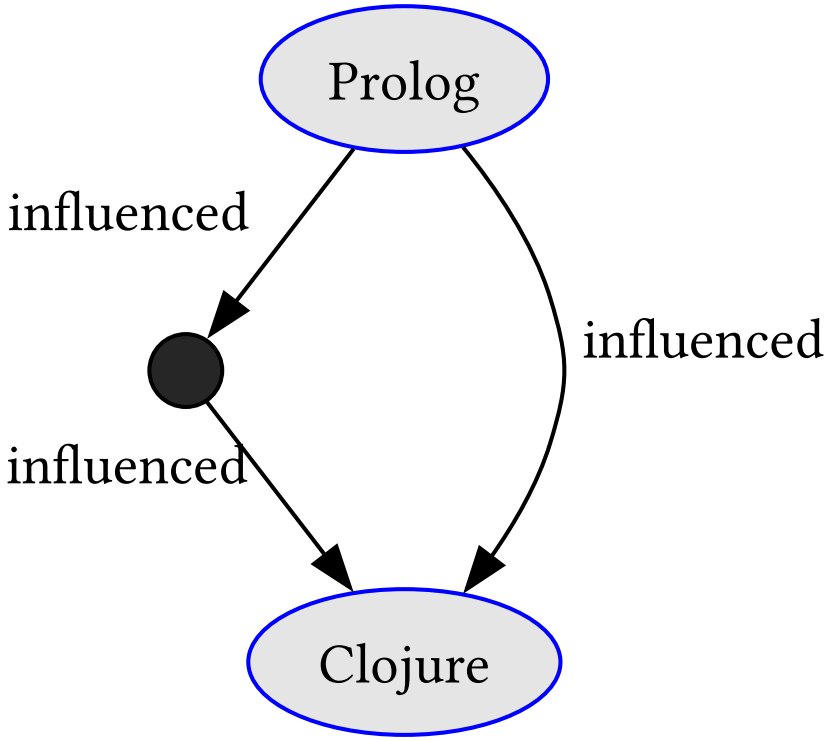
There are 3 kinds of resources:

- IRIs (links)
- Literals (strings, numbers, dates, …)
- Blank Nodes
  - dummy nodes
  - “existentialy quantified” nodes

Resources

There are 3 kinds of resources:

- IRIs (links)
- Literals (strings, numbers, dates, …)
- Blank Nodes
  - dummy nodes
  - “existentialy quantified” nodes



IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs
any ground term	`lang(prolog)`	more flexibility	unification (deep terms)

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs
any ground term	`lang(prolog)`	more flexibility	unification (deep terms)
`iri(IRI)`	`iri('https://www.wikipedia.org/wiki/Prolog')`	pattern matching (IRI checking)	longer terms (hard to read)

IRI representation

Representation	Examples	Good	Bad
IRI atoms	`'https://www.wikipedia.org/wiki/Prolog'`	unification	long atoms (hard to read)
any atoms	`prolog`	readability, unification	extern mapping for IRIs
any ground term	`lang(prolog)`	more flexibility	unification (deep terms)
`iri(IRI)`	`iri('https://www.wikipedia.org/wiki/Prolog')`	pattern matching (IRI checking)	longer terms (hard to read)
`Ns:Frag` and `:(Frag)`	`wiki:prolog`, `:(prolog)`	readability	Namespaces and IRI collision

Blank node representation

Representation	Examples	Comment
atoms starting with `'_:'`	`'_:a'`	unification

Blank node representation

Representation	Examples	Comment
atoms starting with `'_:'`	`'_:a'`	unification
no support (use IRIs)		IRI generation (skolemization)

Blank node representation

Representation	Examples	Comment
atoms starting with `'_:'`	`'_:a'`	unification
no support (use IRIs)		IRI generation (skolemization)
`blank(Labeled, Name)`	`blank(labeled, '_:a')`, `blank(unlabeled, foo(bar, baz))`	more flexible, pattern matching

Literal representation

Representation	Examples	Comment
literal(Lit)	`literal(1)`, `literal(@("prolog", "en"))`	close to prolog semantics

Literal representation

Representation	Examples	Comment
literal(Lit)	`literal(1)`, `literal(@("prolog", "en"))`	close to prolog semantics
literal(Type, Repr) ^^(Repr, Type)	`literal(IntegerIRI, "1")`, `literal(IntegerIRI, "01")`, `literal(LangStrIRI, @("prolog", "en"))`	closer to RDF semantics

Using IRI atoms representation:

```
1 IntegerIRI = 'http://www.w3.org/2001/XMLSchema#integer',
2 LangStrIRI = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#langString'.
```

A RDF Graph is a set of triples

Interface	Examples
assert/retract (implicit graph)	`assert_axiom(prolog, influenced, datalog).` `retract_axiom(prolog, influenced, datalog).`

A RDF Graph is a set of triples

Interface	Examples
assert/retract (implicit graph)	`assert_axiom(prolog, influenced, datalog).` `retract_axiom(prolog, influenced, datalog).`
custom datatype (explicit graph)	`put_axiom(prolog, influenced, datalog, G0, G).` `del_axiom(prolog, influenced, datalog, G0, G).`

Explicit graph allows:

- working with multiple graphs
- set operations (union, intersection, minus, ...)

Queries

Interface	Example
inline queries	`?- Pred = influenced, rdf(prolog, Pred, X), rdf(X, Pred, Y).`

Queries

Interface	Example
inline queries	`?- Pred = influenced, rdf(prolog, Pred, X), rdf(X, Pred, Y).`
Domain Specific Language (similar to DCGs)	`?- Pred = influenced, query(( rdf(prolog, Pred, X), rdf(X, Pred, Y) )).`

DSL helps to achieve:

- load-time optimizations (`term_expansion/2` and `goal_expansion/2`)
- SPARQL translation (to and from)
  - federation queries

About My Library and Me

About My Web Semantic Experience

Semantic Web Course (2025 March $\sim$ July)

- Late-undergraduate/Graduate level
- I wrote the Library in less than 3 weeks (final assignment)
  - unordered list of triples for the triple store
  - most of the implementation relies on `library(lists)` and `library(reif)`
- As far as I know, there are 0 users

About My Web Semantic Experience

Semantic Web Course (2025 March ∼ July)

- Late-undergraduate/Graduate level
- I wrote the Library in less than 3 weeks (final assignment)
  - unordered list of triples for the triple store
  - most of the implementation relies on `library(lists)` and `library(reif)`
- As far as I know, there are 0 users

I have a static Turtle generator (unrelated with my library)

- `github.com/Kiyoshi364/static-memory` (my personal "web site")

About My Web Semantic Experience

Semantic Web Course (2025 March ∼ July)

- Late-undergraduate/Graduate level
- I wrote the Library in less than 3 weeks (final assignment)
  - unordered list of triples for the triple store
  - most of the implementation relies on `library(lists)` and `library(reif)`
- As far as I know, there are 0 users

I have a static Turtle generator (unrelated with my library)

- `github.com/Kiyoshi364/static-memory` (my personal "web site")

Reference links for Semantic Web:

- `github.com/semantalytics/awesome-semantic-web`

About Me

Graduate Student at UFRJ (Brazil)

- My research field is in Programming Languages
- Formalizing connections between Applicative and Concatenative Tacit Programming
  - Combinatory Logic and Concatenative Calculus
  - Publications: `github.com/Kiyoshi364/static-memory`

About Me

Graduate Student at UFRJ (Brazil)

- My research field is in Programming Languages
- Formalizing connections between Applicative and Concatenative Tacit Programming
  - Combinatory Logic and Concatenative Calculus
  - Publications: `github.com/Kiyoshi364/static-memory`

Research interests:

- alternative programming paradigms and computing models
- theorem proving and proof assistants
- static analysis/type systems/logic systems
- creating and using models
- “point at two things and saying ‘they are equal!’”