The N<A>F Design Pattern

Bridging the Gap Between Formal Languages and Natural Languages with Zippers

Sébastien Ferré
Team LIS, Data and Knowledge Management, IRISA/Univ. Rennes 1

Extended Semantic Web Conference (ESWC)
2 June 2016, Heraklion, Crete, Greece

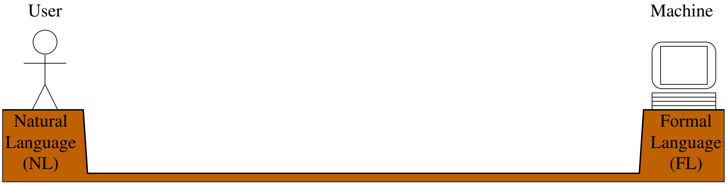
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

UMR IRISA

Overview

1. The Language Gap
2. Principles of the N<A>F Design Pattern
3. Illustration on a Core RDF Query Language
4. Application to 3 Semantic Web Tasks
5. Conclusion

The Gap between Formal Languages and Natural Languages



- Humans speak English, French, Chinese, ...
  Natural Languages (NL)
- Machines speak RDF, OWL, SPARQL, ...
  Formal Languages (FL)
- Only a few humans speak both

Different Kinds of Bridges

Different approaches have been explored to cross the gap for search:

- Question Answering (QA): "unsafe full-way bridge"
  - users express questions in spontaneous NL
  - systems often fail to understand the question or cannot answer it
- Controlled Natural Languages (CNL): "safe half-way bridge"
  - users must use restricted grammar and lexicon
  - systems can help write well-formed questions
- Query Builders (QB): "safe and assisted climbing"
  - users still have to build formal queries
  - systems help build well-formed queries

They offer different trade-offs between expressivity (FL coverage), safeness (reliability), and readability (closeness to NL).

Different Kinds of Bridges

Different approaches have been explored to cross the gap for search:

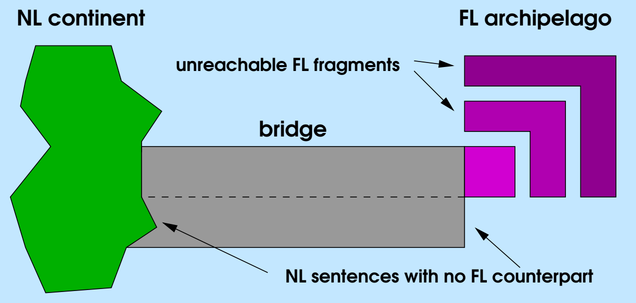
- Question Answering (QA): "unsafe full-way bridge"
  - users express questions in spontaneous NL
  - systems often fail to understand the question or cannot answer it
- Controlled Natural Languages (CNL): "safe half-way bridge"
  - users must use restricted grammar and lexicon
  - systems can help write well-formed questions
- Query Builders (QB): "safe and assisted climbing"
  - users still have to build formal queries
  - systems help build well-formed queries

They offer different trade-offs between expressivity (FL coverage), safeness (reliability), and readability (closeness to NL).

The Problem of Adequacy

Adequacy = expressivity + safeness

- *an essential property of language bridges*
- expressivity (∼recall): proportion of FL sentences reachable through the bridge
- safeness (∼precision): proportion of paths on the bridge leading to correct FL sentences



The N<A>F Design Pattern

A design pattern to bridge the gap between FL and NL

Design Pattern

In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. It is a description or template for how to solve a problem that can be used in many different situations. [Wikipedia]

Compare: (AST = Abstract Syntax Tree = intermediate representation)

- QA, CNL: written NL $\longrightarrow$ AST $\longrightarrow$ FL
- QB: incrementally built FL
- N<A>F: NL $\longleftarrow$ incrementally built AST $\longrightarrow$ FL
  $\Rightarrow$ N<A>F produces "safe and full-way bridges"

Pros and Cons

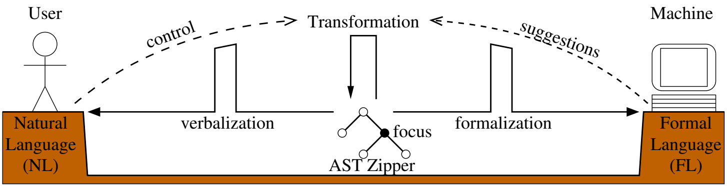
PROS

1. bridges the NL-FL gap because two-way synchronous translations
2. scales in expressivity because ambiguities are solved piecewise during building
3. ensures strong safeness because fine-grained guidance during building
4. offers a lot of flexibility because building applies to a tree, not a sequence of words
5. applies to various tasks because no assumption is made on the FL

CONS

1. does not apply to spontaneous NL or existing texts
2. has slower interaction because of the building process

Bridging the Gap with Zippers



A kind of “suspended bridge”:

- **pillar** Abstract Syntax Trees (AST) + Huet’s zippers for focus
- **suspender** transformations of AST zippers
- **decks** translations defined as Montague grammars
- **cables** system suggestions and user control

Illustration on a Core RDF Query Language (CRQL)

To show a concrete application of the N<A>F design pattern

- task: semantic search on RDF data
- formal language: CRQL, a fragment of SPARQL
  tree patterns, simple filters, unions, negations
- safeness criteria: avoid empty results

Bridge components:

1. ASTs
2. AST zippers for focus representation
3. AST zipper transformations for AST building
4. translation to SPARQL
5. translation to English
6. computation of suggestions

Illustration on a Core RDF Query Language (CRQL)

To show a concrete application of the N<A>F design pattern

- task: semantic search on RDF data
- formal language: CRQL, a fragment of SPARQL
  tree patterns, simple filters, unions, negations
- safeness criteria: avoid empty results

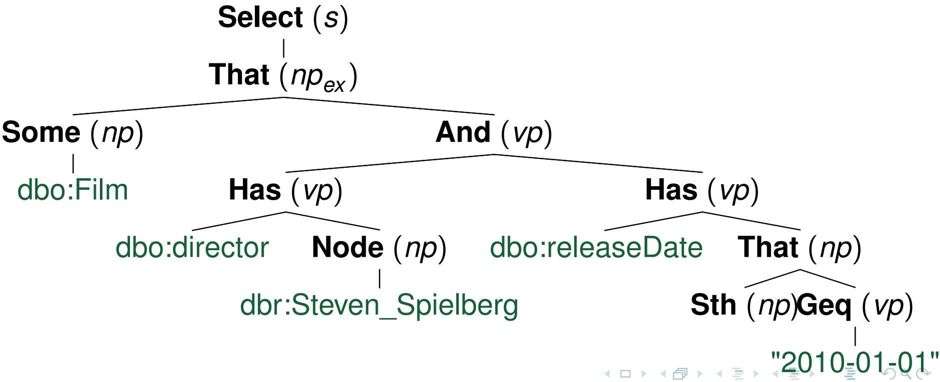
Bridge components:

1. ASTs
2. AST zippers for focus representation
3. AST zipper transformations for AST building
4. translation to SPARQL
5. translation to English
6. computation of suggestions

1. CRQL ASTs

ASTs are close to NL syntax *but* much more abstract

- sentences (*s*) denote queries
- noun phrases (*np*) denote sets of entities
- verb phrases (*vp*) denote constraints on entities
- words are RDF classes, properties, and nodes



1. ASTs Specification

ASTs are trees that can be specified with algebraic datatypes*:

$$
\begin{array}{lcl}
s & ::= & \textbf{Select}(np) \\
\\
np & ::= & \textbf{Something} \\
 & | & \textbf{Some}(class) \\
 & | & \textbf{Node}(node) \\
 & | & \textbf{That}(np, vp) \\
\\
vp & ::= & \textbf{IsA}(class) \\
 & | & \textbf{Has}(prop, np) \\
 & | & \textbf{IsOf}(prop, np) \\
 & | & \textbf{Geq}(lit) \\
 & | & \textbf{True} \\
 & | & \textbf{And}(vp, vp) \mid \textbf{Or}(vp, vp) \mid \textbf{Not}(vp)
\end{array}
$$

∗ *source code online in OO style (Java) and ML style (OCaml)*

2. AST Zippers

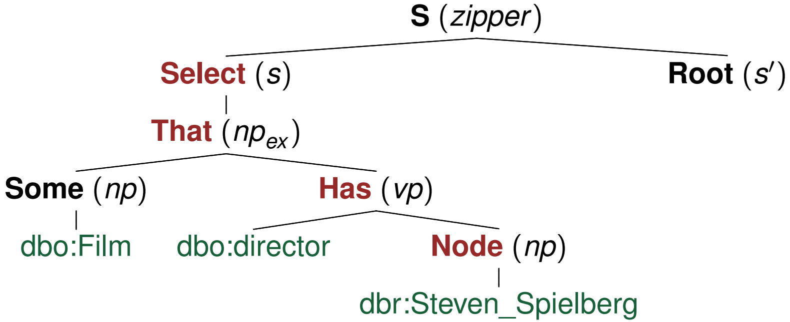
Huet's Zippers (*functional pearl at J. Functional Prog., 1997*)

- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context

2. AST Zippers

Huet's Zippers (*functional pearl at J. Functional Prog., 1997*)

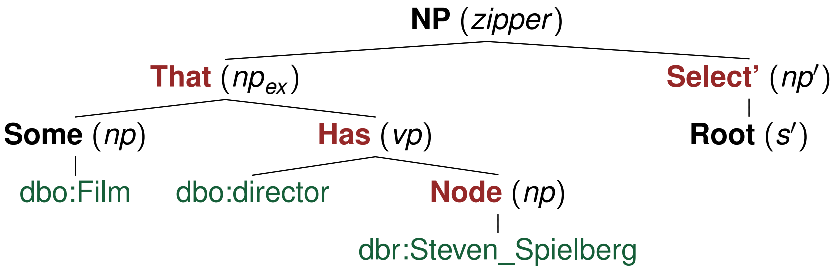
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



2. AST Zippers

Huet's Zippers (*functional pearl at J. Functional Prog., 1997*)

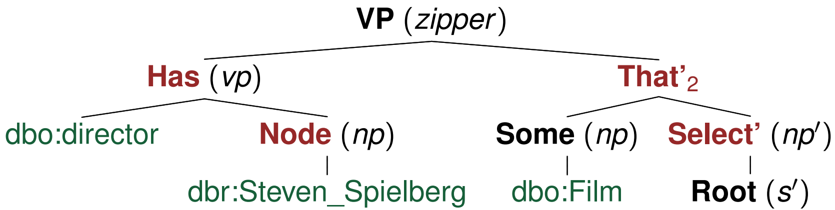
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



2. AST Zippers

Huet's Zippers (*functional pearl at J. Functional Prog., 1997*)

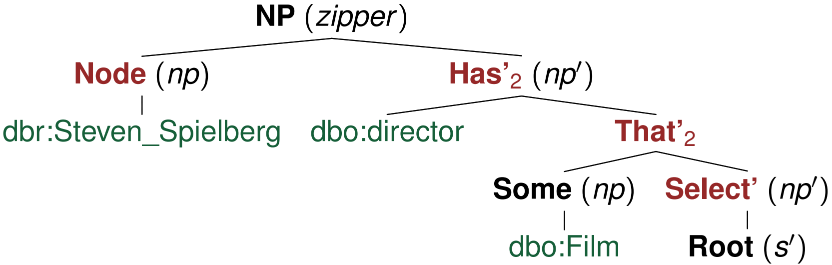
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



2. AST Zippers

Huet's Zippers (*functional pearl at J. Functional Prog., 1997*)

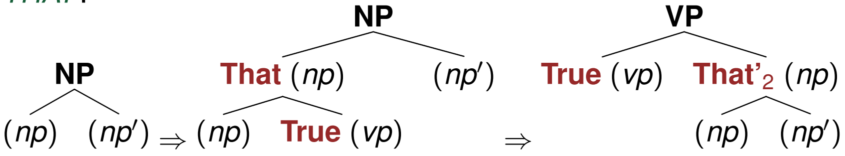
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



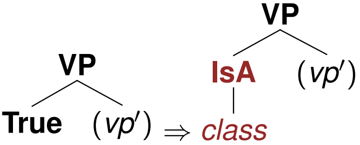
3. AST Zipper Transformations

A transformation goes from zipper to zipper, used as a building step

- *THAT*:



- *INSERT*(**IsA**(*class*)):



- *AND*, *OR*, *NOT*: algebraic operators, similar to *THAT*
- *DOWN*, *UP*, *LEFT*, *RIGHT*: focus moves

Theorem

The set of transformations is CRQL-complete from **Select**(**Sth**).

4. Translation to SPARQL (Formalization)

R. Montague's Grammar (*"English as a formal language", 1970*)

- designed for translation from NL to logic
- compositional semantics based on lambda calculus
- Montague grammar = grammar rules + lambda-terms
  - *here, AST datatypes play the role of grammars*

Exerpt

$vp ::= \mathbf{IsA}(class) \qquad \lambda x.(x + \text{'}\texttt{a}\text{'} + class)$

$vp ::= \mathbf{Not}(vp_1) \qquad \lambda x.(\text{'}\texttt{FILTER NOT EXISTS \{}\text{'} + (vp_1\ x) + \text{'}\texttt{\}}\text{'})$

4. Full Montague Grammar for Formalization in SPARQL

s	$::=$	**Select**(np)	$\texttt{'SELECT ?x}_1\texttt{... WHERE \{'} + (np\ \lambda x.(\texttt{''})) + \texttt{'\}'}$
np	$::=$	**Something**$_i$	$\lambda d.((d\ \texttt{'?x}_i\texttt{'}))$
	$\mid$	**Some**($class$)$_i$	$\lambda d.(\texttt{'?x}_i\texttt{ a'} + class + \texttt{' .'} + (d\ \texttt{'?x}_i\texttt{'}))$
	$\mid$	**Node**($node$)	$\lambda d.((d\ node))$
	$\mid$	**That**(np, vp)	$\lambda d.(np\ \lambda x.((d\ x) + \texttt{' .'} + (vp\ x)))$
vp	$::=$	**IsA**($class$)	$\lambda x.(x + \texttt{' a'} + class)$
	$\mid$	**Has**($prop, np$)	$\lambda x.((np\ \lambda y.(x + prop + y)))$
	$\mid$	**IsOf**($prop, np$)	$\lambda x.((np\ \lambda y.(y + prop + x)))$
	$\mid$	**Geq**(lit)	$\lambda x.(\texttt{'FILTER('} + x + \texttt{'>='} + lit + \texttt{')'})$
	$\mid$	**True**	$\lambda x.(\texttt{''})$
	$\mid$	**And**(vp_1, vp_2)	$\lambda x.((vp_1\ x) + \texttt{' .'} + (vp_2\ x))$
	$\mid$	**Or**(vp_1, vp_2)	$\lambda x.(\texttt{'\{'} + (vp_1\ x) + \texttt{'\} UNION \{'} + (vp_2\ x) + \texttt{'\}'})$
	$\mid$	**Not**(vp)	$\lambda x.(\texttt{'FILTER NOT EXISTS \{'} + (vp\ x) + \texttt{'\}'})$

5. Translation to English (Verbalization)

Montague grammars can also be used here

- English as target language
- compositional generation of NL phrases
  - s ⇝ sentences, np ⇝ noun phrases
  - vp ⇝ relative clauses parametrized by negation ($\lambda n.$)
  - $class$, $prop$ ⇝ noun

Excerpt

s	$::=$	**Select**(np)	$\texttt{'Give me'} + np$
np	$::=$	**That**(np, vp)	$np + (vp\ 0)$
vp	$::=$	**IsA**($class$)	$\lambda n.(\texttt{'that'} + (is\ n) + \texttt{'a(n)'} + class)$
	$\mid$	**Has**($prop, np$)	$\lambda n.(\texttt{'whose'} + prop + (is\ n) + np)$
	$\mid$	**Not**(vp)	$\lambda n.(vp\ \overline{n})$

$is\ 0 = \texttt{'is'}$

$is\ 1 = \texttt{'is not'}$

4 & 5.Translation Example

The example AST above has the following translations.

SPARQL

```
SELECT ?x1 ?x2 WHERE
{ ?x1 a dbo:Film .
  ?x1 dbo:director dbr:Steven_Spielberg .
  ?x1 dbo:releaseDate ?x2 . FILTER (?x2 >= "2010-01-01") }
```

English

Give me a film
whose director is Steven Spielberg
and whose release date is after January 1st, 2010

6. Computation of System Suggestions

No general technique for this component:

- depends on the task
- depends on the FL semantics
- depends on the safeness criteria

For semantic search with CRQL, suggestions are computed from SPARQL results and from the focus entity `x`

- nodes: values of `x`
- classes: values of `?c` s.t. `{ x a ?c }`
- properties: values of `?p` s.t. `{ x ?p [] }` or `{ [] ?p x }`

Theorem

The suggestions prevent empty results (safeness), and yet are complete w.r.t. non-empty CRQL queries (expressivity)
$\Rightarrow$ perfect adequacy to CRQL.

6. Computation of System Suggestions

No general technique for this component:

- depends on the task
- depends on the FL semantics
- depends on the safeness criteria

For semantic search with CRQL, suggestions are computed from SPARQL results and from the focus entity `x`

- nodes: values of `x`
- classes: values of `?c` s.t. `{ x a ?c }`
- properties: values of `?p` s.t. `{ x ?p [] }` or `{ [] ?p x }`

Theorem

The suggestions prevent empty results (safeness), and yet are complete w.r.t. non-empty CRQL queries (expressivity)
$\Rightarrow$ perfect adequacy to CRQL.

Application to 3 Semantic Web Tasks

To show the effectiveness and genericity of the N<A>F design pattern

	SPARKLIS	SEWELIS/UTILIS	PEW
task	querying RDF endpoints	authoring RDF descriptions	completing OWL ontologies
FL	SPARQL	RDF	OWL
expres-sivity	CRQL + cyclic patterns + OPTIONAL + ordering + aggregations	conjunctive sub-set of CRQL	CRQL - filter - non-atomic neg-ations
safe-ness	no empty results	similarity to previ-ous descriptions	no inconsistency
sugges-tions	SPARQL eval.	query relaxation	satisfiability checks

Scaling in Expressivity

The N<A>F design pattern has proved valuable for increasing FL coverage in a modular way

- SPARKLIS: RDF analytics (see QALD-6 challenge)
  - expressions, nested aggregations
  - Give me the average total budget of films per director
- SEWELIS/UTILIS: update rules
  - quantifiers “every” and “no”
  - Every film produced in USA is also produced in the United States

Conclusion

The N<A>F design patter is

1. a powerful strategy to build bridges over the NL-FL gap
   - users are never exposed to FL
   - and machines are never exposed to NL
   - users cannot fall in the gap (safeness)
   - large subsets of FL are reachable by users (expressivity)
2. an interesting alternative to Question Answering
   - that avoids the hard problem of NL understanding
   - that scales in expressivity in a modular way
   - that applies to various tasks and FL

The End

Questions ?