Responsive and Flexible CNL Authoring with Zipper-based Transformations

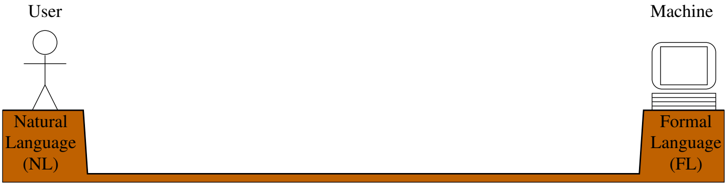
Sébastien Ferré
Team SemLIS, Data and Knowledge Management, IRISA/Univ. Rennes 1

Controlled Natural Languages (CNL)
27 August 2018, Maynooth, Ireland

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

Overview

The Gap between Formal Languages and Natural Languages

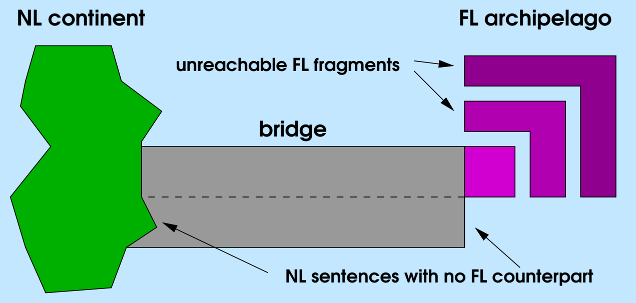


- Humans speak English, French, Chinese, ...
  Natural Languages (NL)
- Machines speak RDF, OWL, SPARQL, ...
  Formal Languages (FL)
- Only a few humans speak both...
  ... so we need bridges over the gap

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



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
  - low coverage of target FL
- Controlled Natural Languages (CNL): "safe half-way bridge"
  - wide coverage of target FL
  - users must use restricted grammar and lexicon
  - systems can help write well-formed questions (autocompletion)
- 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
  - low coverage of target FL
- Controlled Natural Languages (CNL): "safe half-way bridge"
  - wide coverage of target FL
  - users must use restricted grammar and lexicon
  - systems can help write well-formed questions (autocompletion)
- 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).

Limits of Autocompletion for CNL Authoring

Autocompletion

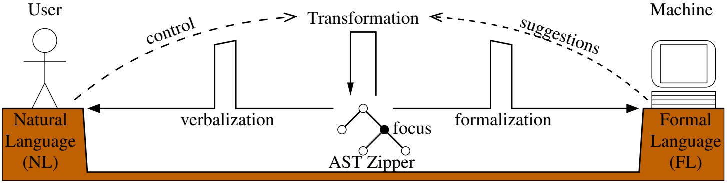
Suggest the next possible words according to the grammar and lexicon.

Limits

- Responsiveness
  - partial sentence at most steps
  - hence no translation/interpretation in FL
  - hence lack of feedback: e.g. query results
- Flexibility
  - in general, only adding words at the end
  - sometimes, one word at a time
  - restricted edition compared to text editors (cursor)

Bridging the Gap with Zippers

The N<A>F design pattern for responsive and flexible CNL authoring.



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

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. is responsive because a complete sentence is defined at all steps
5. offers a lot of flexibility because building applies to a tree, not a sequence of words, and focus as cursor
6. 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

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, 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, unions, negations
- safeness criteria: avoid empty results

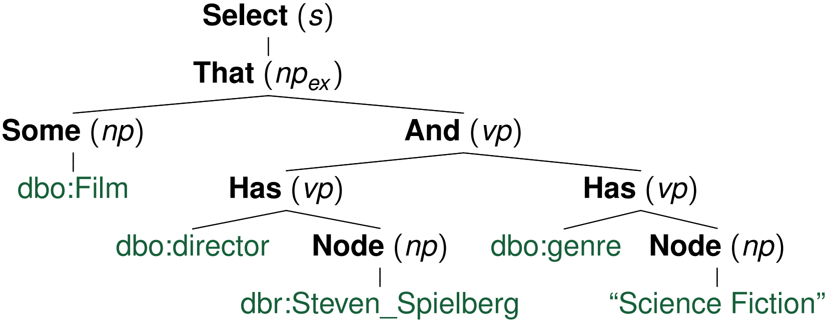
Bridge components:

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 conditions on entities
- words are RDF classes, properties, and nodes



1. ASTs Specification

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

$$
\begin{array}{lcl}
s & := & \textbf{Select}(np) \\
 & & \\
np & := & \textbf{Node}(node) \\
 & | & \textbf{DetThat}(det, class, vp) \\
 & | & \textbf{And}(np, np) \mid \textbf{Or}(np, np) \mid \textbf{Not}(np) \\
 & & \\
det & := & \textbf{Some} \mid \textbf{Every} \mid \textbf{No} \\
 & & \\
vp & := & \textbf{IsA}(class) \\
 & | & \textbf{Has}(prop, np) \\
 & | & \textbf{IsOf}(prop, np) \\
 & | & \textbf{True} \\
 & | & \textbf{And}(vp, vp) \mid \textbf{Or}(vp, vp) \mid \textbf{Not}(vp)
\end{array}
$$

∗ *source code online in ML style (OCaml)*

2. AST Zippers

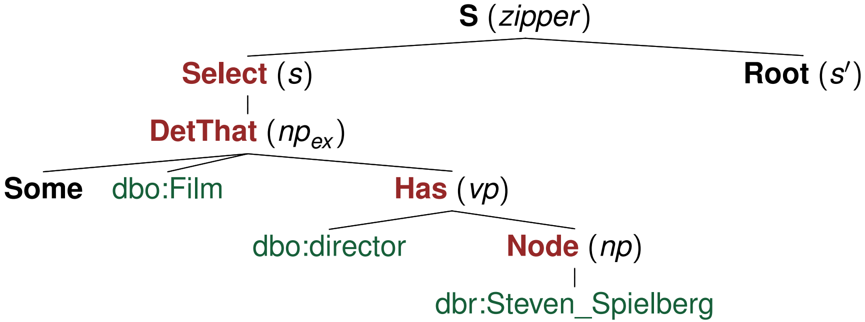
Huet's Zippers (*functional pearl at J. Functional Prog., 1997*)

- 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*)

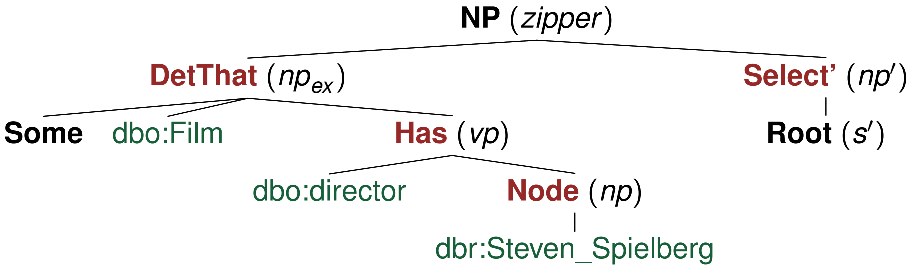
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



2. AST Zippers

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

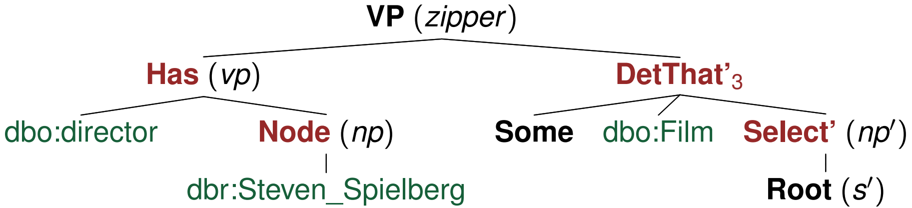
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



2. AST Zippers

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

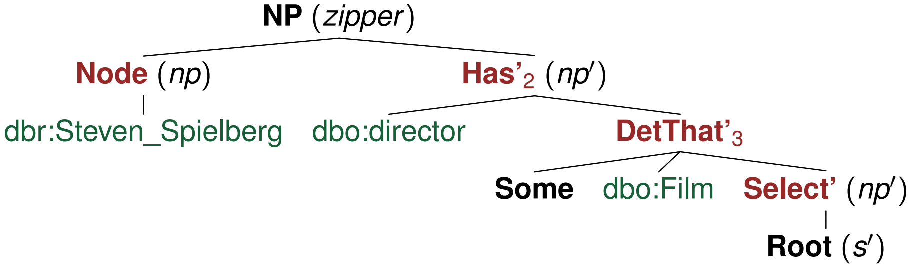
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



2. AST Zippers

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

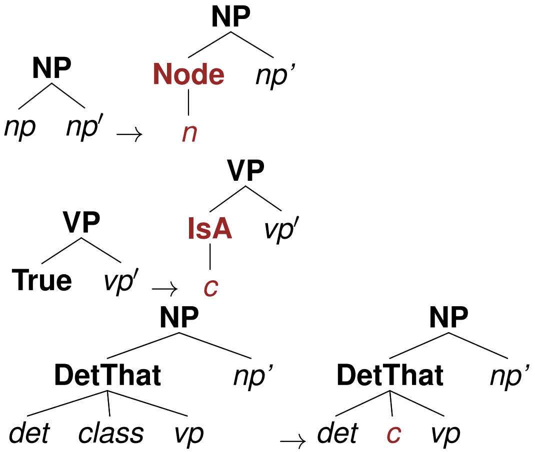
- type-safe representation of focus in complex data structures
- efficient focus-centered edition of data structures (transformations)
- *open and close data structures like a jacket!*
- s', np', vp' are datatypes for the contexts of s, np, vp
- zipper = sub-tree under focus + context



3. AST Zipper Transformations (1/3)

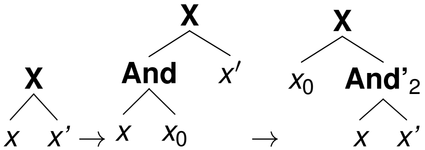
A transformation goes from zipper to zipper, used as a building step

- *NODE*(n):
- *CLASS*(c):



- *DET*(d), *PROP*(p): insertions, similar to *NODE* and *CLASS*

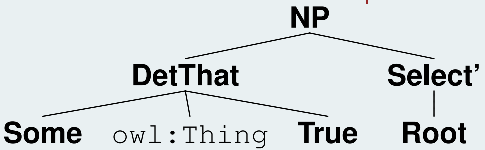
3. AST Zipper Transformations (2/3)

- *AND*: 

(x_0 = default x)
- *OR*, *NOT*: algebraic operators, similar to *AND*
- *DOWN*, *UP*, *LEFT*, *RIGHT*: focus moves

Theorem

The set of transformations is CRQL-complete from initial zipper



3. AST Zipper Transformations (3/3)

The above example AST is reached by the following sequence of transformations:

1. CLASS(dbo:film): **NP**(**DetThat**(**Some**,dbo:Film,**True**), **Select'**(**Root**))
2. DOWN: **VP**(**True**, **DetThat'**$_3$(**Some**,dbo:Film,**Select'**(**Root**)))
3. PROP(dbo:director): **VP**(**Has**(dbo:director,**DetThat**(**Some**,owl:Thing,**True**)), **DetThat'**$_3$(**Some**,dbo:Film,**Select'**(**Root**)))
4. DOWN: **NP**(**DetThat**(**Some**,owl:Thing,**True**), **Has'**$_2$(dbo:director, **DetThat'**$_3$(**Some**,dbo:Film,**Select'**(**Root**)))
5. NODE(dbr:Steven_Spielberg): **NP**(**Node**(dbr:Steven_Spielberg), **Has'**$_2$(dbo:director, **DetThat'**$_3$(**Some**,dbo:Film,**Select'**(**Root**))))
6. UP, UP, ...: **S**(**Select**(**DetThat**(**Some**,dbo:Film,**Has**(dbo:director,**Node**(dbr:Steven_Spielberg)))) **Root**)

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

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

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

4. Full Montague Grammar for Formalization in SPARQL

s	$:=$	**Select**(np)	$'\texttt{SELECT ?x}_1\texttt{... WHERE \{}' + (np\ \lambda x.('')) + '\ \texttt{\}}$
np	$:=$	**Node**($node$)	$\lambda d.((d\ node))$
	$\mid$	**DetThat**(det, cl, vp)	$\lambda d.(det\ cl\ \lambda x.((d\ x) + '\ \texttt{.}' + (vp\ x)))$
	$\mid$	**And**(np_1, np_2)	$\lambda d.((np_1\ d) + '\ \texttt{.}' + (np_2\ d))$
	$\mid$	**Or**(np_1, np_2)	$\lambda d.('\texttt{\{}' + (np_1\ d) + '\ \texttt{\} UNION \{}' + (np_2\ d) + '\ \texttt{\}}$
	$\mid$	**Not**(np)	$\lambda d.('\texttt{NOT \{}' + (np\ d) + '\ \texttt{\}}')$
det	$:=$	**Some**	$\lambda d_1.\lambda d_2.((d_1\ '\texttt{?x}_i') + '\ \texttt{.}' + (d_2\ '\texttt{?x}_i'))$
	$\mid$	**No**	$\lambda d_1.\lambda d_2.('\texttt{NOT \{}' + (d_1\ '\texttt{?x}_i' + '\ \texttt{.}' + (d_2\ '\texttt{?x}_i'$
	$\mid$	**Every**	$\lambda d_1.\lambda d_2.('\texttt{NOT \{}' + (d_1\ '\texttt{?x}_i' + '\texttt{. NOT\{}' + (d_2$
vp	$:=$	**IsA**($class$)	$\lambda x.(x + '\ \texttt{a}' + class)$
	$\mid$	**Has**($prop, np$)	$\lambda x.((np\ \lambda y.(x + prop + y)))$
	$\mid$	**IsOf**($prop, np$)	$\lambda x.((np\ \lambda y.(y + prop + x)))$
	$\mid$	**True**	$\lambda x.('')$
	$\mid$	**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)	$\lambda x.('\texttt{NOT \{}' + (vp\ x) + '\ \texttt{\}}')$

5. Translation to English (Verbalization)

Montague grammars can also be used here

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

Excerpt

$$
\begin{array}{lcll}
s & := & \textbf{Select}(np) & \texttt{'Give me'} + np \\
np & := & \textbf{DetThat}(det, c, vp) & det + c + (vp\ 0) \\
vp & := & \textbf{IsA}(class) & \lambda n.(\texttt{'that'} + (is\ n) + \texttt{'a(n)'} + class) \\
 & | & \textbf{Has}(prop, np) & \lambda n.(\texttt{'whose'} + prop + (is\ n) + np) \\
 & | & \textbf{Not}(vp) & \lambda n.(vp\ \overline{n})
\end{array}
$$

$$
\begin{array}{lcl}
is\ 0 & = & \texttt{'is'} \\
is\ 1 & = & \texttt{'is not'}
\end{array}
$$

4 & 5.Translation Example

The example AST above has the following translations.

SPARQL

```
SELECT ?x1 WHERE
{ ?x1 a dbo:Film .
  ?x1 dbo:director dbr:Steven_Spielberg . }
```

English

Give me a film
whose director is Steven Spielberg

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, suggested *insertion* transformations 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, suggested *insertion* transformations 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 + expressions	conjunctive sub-set of CRQL	CRQL - non-atomic neg-ations
safe-ness	no empty results	similarity to previ-ous descriptions	no inconsistency
sugges-tions	SPARQL eval.	query relaxation	satisfiability checks

Results from User Studies

- SPARKLIS has 200-2000 hits per month since Spring 2014
- SPARKLIS has been adopted by two French institutions
- UTILIS' fine-grained suggestions prefered to Protégé's
- UTILIS help to produce more consistent data
- PEW better in quantity and quality than Protégé
  - 56% vs 24% completion in formalization of hand anatomy
  - more axioms produced with smaller error rate
- Main difficulty: understand the focus, its impact on suggestions, and the need to move it

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 (readability)
   - 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 CNL Autocompletion
   - formal interpretation (e.g. results) is available at all steps (responsiveness)
   - query elements can be inserted/deleted at any focus (flexibility)
   - edition steps are more semantic
     e.g. inserting a property means crossing a relation in the RDF graph
3. 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 ?