FROM **CONTROLLED NATURAL LANGUAGE** TO **ANSWER SET PROGRAMS** AND BACK AGAIN VIA A **BI-DIRECTIONAL GRAMMAR**

BASED ON: "*SPECIFYING AND VERBALISING ANSWER SET PROGRAMS IN CONTROLLED NATURAL LANGUAGE*" [R. SCHWITTER, 2018]

Eugenia Soroka

Ph.D. student

Computer Science Dept.

Stony Brook University

Fall 2018 – Stony Brook, NY

MOTIVATION

There exist a number of controlled natural languages that have been designed as high-level interface languages to knowledge systems. However, **none of the underlying grammars of these CNLs is <u>bi-directional</u>** in the sense that a specification can be written in controlled natural language, translated into a formal target representation, and then - after potential modifications of that target representation - back again into the same subset of natural language.

SYSTEM INPUT AND OUTPUT

1. **Input sentences:** written in CNL by a user.
2. **Internal representation:** special internal format to represent input sentences as logic rules or clauses.
3. **ASP representation:** internal representation transformed into ASP format (e.g. suitable for *clingo*).
4. **Generated (back) sentences:** sentences reconstructed from internal representation, in ideal case – the same as input sentences.

SYSTEM COMPONENTS

1. **Lexicon** – vocabulary of allowed words
2. **Bi-directional Definite Clause Grammar (DCG)**
3. **Rules for processing** – to convert sentences to internal representation
4. **Rules for converting to ASP** – to convert internal representation to ASP
5. **Rules for generation** – to convert internal representation back to sentences

SYSTEM COMPONENTS

1. Lexicon example:

```
lexicon(cat:noun, wform:[penguin], arg:X, term:penguin(X)).
lexicon(cat:noun, wform:[bird],    arg:X, term:bird(X)).
lexicon(cat:noun, wform:[eagle],   arg:X, term:eagle(X)).
lexicon(cat:iv,   wform:[fly],     arg:X, term:fly(X)).
...
lex(arg:[emperor]).
lex(arg:[baldey]).
...
```

Vocabulary words

Agents

SYSTEM COMPONENTS

2. Bi-directional Definite Clause Grammar (DCG):

```
s --> np, vp, ['.'].

np --> det, noun.

np --> noun.

vp --> iv.

...
```

```
s(mode:Mode, sem:M1-M2, st:St) -->
    np(mode:Mode, arg:X, sco:S, sem:M1-M2),
    vp(mode:Mode, arg:X, sem:S, st:St), ['.'].

np(mode:Mode, arg:X, sco:S, sem:M1-M2) -->
    det(mode:Mode, arg:X, res:R, sco:S, sem:M1-M2),
    noun(mode:Mode, arg:X, sem:R, st:pos).
np(mode:Mode, arg:X, sem:S, st:St) -->
    noun(mode:Mode, arg:X, sem:S, st:St).

vp(mode:Mode, arg:X, sem:M, st:St) -->
    iv(mode:Mode, arg:X, sem:M, st:St).
...
```

SYSTEM COMPONENTS

3. **Rules for processing:**

```
s(mode:proc, sem:M1-M2, st:St) -->
  np(mode:proc, arg:X, sem:M1-M0, st:pos),
  vp(mode:proc, arg:X, sem:M0-M2, st:St),
  ['.'].
...
noun(mode:proc, arg:X, sem:[M1|M2]-[[T|M1]|M2], st:pos) -->
   { lexicon(cat:noun, wform:List, arg:X, term:T);
     (   lex(arg:[X]),  List = [X]   )},
   List.
noun(mode:proc, arg:X, sem:[M1|M2]-[[-T|M1]|M2], st:neg) -->
   { lexicon(cat:noun, wform:List, arg:X, term:T);
     (   lex(arg:[X]),  List = [X]   )},
   List.
```

SYSTEM COMPONENTS

4. **Rules for converting to ASP:**

```
make_ASP([]).

make_ASP([T|Clauses]) :-
   write(T), writeln(.),
   make_ASP(Clauses).

make_ASP([forall(_,[T1]==>[T2])|Clauses]) :-
   portray_clause(T2 :- T1),
   make_ASP(Clauses).
...
```

SYSTEM COMPONENTS

5. Rules for generation:

```
s(mode:gen, sem:S, st:St) -->
   np(mode:gen, arg:X, sem:S, st:pos),
   vp(mode:gen, arg:X, sem:S, st:St),
   ['.'].
...
vp(mode:gen, arg:X, sem:S, st:pos) -->
   ['is'],
   jj(mode:gen, arg:X, sem:S, st:pos).
vp(mode:gen, arg:X, sem:S, st:neg) -->
   ['is not'],
   jj(mode:gen, arg:X, sem:S, st:neg).
...
```

RUNNING THE PROGRAM: **EXAMPLE 1**

1. Input sentences:

```
[baldey,is,eagle,.]
[emperor,is,penguin,.]
[every,penguin,is,bird,.]
[every,eagle,is,bird,.]
[every,penguin,does not,fly,.]
```

2. Internal representation:

```
eagle(baldey)
penguin(emperor)
forall(A,[penguin(A)]==>[bird(A)])
forall(A,[eagle(A)]==>[bird(A)])
forall(A,[penguin(A)]==>[-fly(A)])
```

1

3. ASP representation:

```
eagle(baldey).
penguin(emperor).
bird(A) :- penguin(A).
bird(A) :- eagle(A).
-fly(A) :- penguin(A).
```

2

4. Generated (back) sentences:

```
[baldey,is,eagle,.]
[emperor,is,penguin,.]
[every,penguin,is,bird,.]
[every,eagle,is,bird,.]
[every,penguin,does not,fly,.]
```

SOLVING ASP

clingo version 5.3.0 (web version: https://potassco.org/clingo/run/)

ASP:

```
eagle(baldey).
penguin(emperor).
bird(A) :- penguin(A).
bird(A) :- eagle(A).
-fly(A) :- penguin(A).
```

Solution:

```
Answer: 1
penguin(emperor) bird(baldey) bird(emperor)
eagle(baldey) -fly(emperor)
SATISFIABLE
Models       : 1
```

If we add a rule `fly(A) :- bird(A), not -fly(A).`, then the solution is:

```
penguin(emperor) bird(baldey) bird(emperor) eagle(baldey) -fly(emperor)
fly(baldey)
```

RUNNING THE PROGRAM: **EXAMPLE 2**

1. Input sentences:

```
[alice,is,girl,.]
[every,girl,is,happy,.]
[every,student,is not,happy,.]
[bob,is,student,.]
```

2. Internal representation:

```
girl(alice)
forall(A,[girl(A)]==>[happy(A)])
forall(A,[student(A)]==>[-happy(A)])
student(bob)
```

3. ASP representation:

```
girl(alice).
happy(A) :- girl(A).
-happy(A) :- student(A).
student(bob).
```

4. Generated (back) sentences:

```
[alice,is,girl,.]
[every,girl,is,happy,.]
[every,student,is not,happy,.]
[bob,is,student,.]
```

Solution: `girl(alice) happy(alice) -happy(bob) student(bob)`

EXPERIENCE GAINED

While working on this project, I:

1. Learned how definite clause grammars work, how to write one in Prolog
2. Learned how to utilize natural language processing in Prolog to specify and verbalize answer set programs
3. Came up with specific format and rules for defining different parts of speech
4. Based on the paper, wrote a bi-directional grammar, which can be used for converting CNL sentences to ASP and back

FUTURE WORK

To do next:

1. Add rules for input sentences containing "and/or"
2. Add rules for input sentences containing "have/has"
3. Add more parts of speech (and corresponding grammar rules)
4. Broaden the lexicon