Speech Recognition Grammar Compilation in GF

Björn Bringert
bringert@cs.chalmers.se
Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University

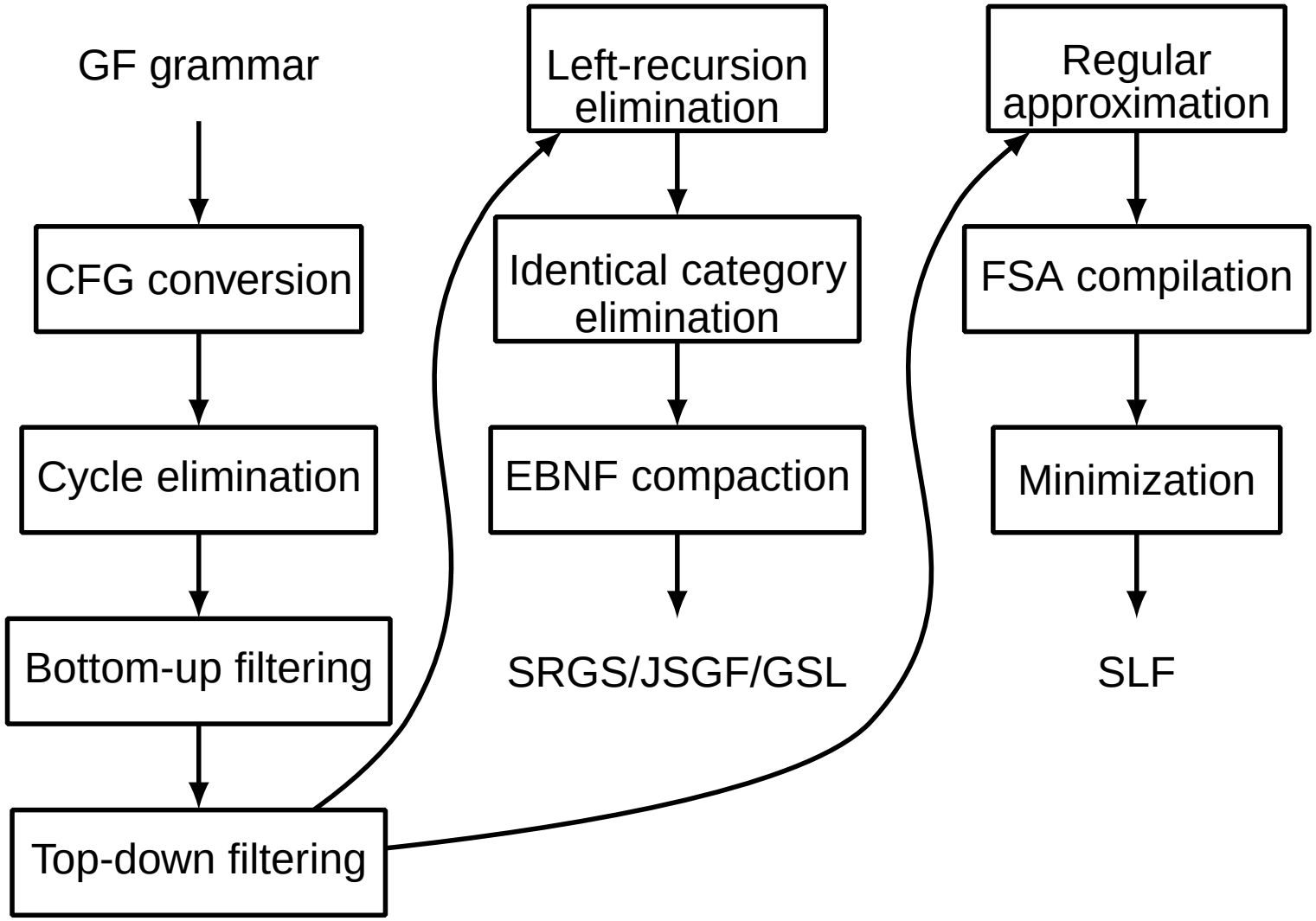
WHY A SPEECH RECOGNITION GRAMMAR COMPILER?

- More expressive.
  - Lower development cost.
  - Higher quality.
- Multiple components based on one grammar.
- Consistency between semantics and language models.
- Multilinguality.
- Portability between speech recognizers.
- Portability across domains.

Features

- GF as source language:
  - Multilingual grammars with shared semantics.
  - Speech recognition, parsing and linearization from a single grammar.
  - GF resource grammar library.
- Many output formats:
  - Context-free: GSL, SRGS (XML, ABNF), JSGF.
  - Regular: SLF, SRGS (XML, ABNF).
- Generates compact grammars.
- Optional embedded semantics (SISR).

COMPILATION PIPELINE



Example abstract syntax

```
abstract Toy0 = {
  flags startcat = NP;
  cat
    NP; Noun; Spec;
  fun
    SpecNoun  : Spec → Noun → NP;
    One, Two  : Spec;
    Felis, Canis : Noun;
    Black     : Noun → Noun;
}
```

Example parameterized concrete syntax

incomplete concrete Toy0I **of** Toy0 =

  open Syntax, Lexicon **in** {

  lincat Spec = Det; Noun = CN; NP = Utt;

  lin

    SpecNoun $spec\ noun = mkUtt\ (mkNP\ spec\ noun)$;

    One $= mkDet\ n1\_Numeral$;

    Two $= mkDet\ n2\_Numeral$;

    Felis $= mkCN\ cat\_N$;

    Canis $= mkCN\ dog\_N$;

    Black $= mkCN\ black\_A$;

}

EXAMPLE CONCRETE SYNTAXES

concrete Toy0Eng **of** Toy0 = Toy0I **with**
  (Syntax = SyntaxEng),
  (Lexicon = LexiconEng) ** {
  flags language = en_US;
}

concrete Toy0Fre **of** Toy0 = Toy0I **with**
  (Syntax = SyntaxFre),
  (Lexicon = LexiconFre) ** {
  flags language = fr_FR;
}

CFG CONVERSION

- ➜ Expands all records, parameters and variants.
- ➜ Explodes.
- ➜ Algorithm by Ljunglöf (2004).

CFG for English example:

```
NP.s → Spec.n=Pl;.s Noun.s!Pl!Nom
NP.s → Spec.n=Sg;.s Noun.s!Sg!Nom
Noun.s!Pl!Acc → "black" Noun.s!Pl!Acc
Noun.s!Pl!Acc → "cats"
Noun.s!Pl!Acc → "dogs"
Noun.s!Pl!Gen → "black" Noun.s!Pl!Gen
Noun.s!Pl!Gen → "cats'"
Noun.s!Pl!Gen → "dogs'"
Noun.s!Pl!Nom → "black" Noun.s!Pl!Nom
Noun.s!Pl!Nom → "cats"
Noun.s!Pl!Nom → "dogs"
```

```
Noun.s!Sg!Acc → "black" Noun.s!Sg!Acc
Noun.s!Sg!Acc → "cat"
Noun.s!Sg!Acc → "dog"
Noun.s!Sg!Gen → "black" Noun.s!Sg!Gen
Noun.s!Sg!Gen → "cat's"
Noun.s!Sg!Gen → "dog's"
Noun.s!Sg!Nom → "black" Noun.s!Sg!Nom
Noun.s!Sg!Nom → "cat"
Noun.s!Sg!Nom → "dog"
Spec.n=Pl;.s → "two"
Spec.n=Sg;.s → "one"
```

CYCLE ELIMINATION

- Removes directly and indirectly cyclic productions.
- Left-recursion elimination can’t handle cycles.
- Cycles are rare in real grammars.

BOTTOM-UP FILTERING

- ➜ Removes productions which produce no finite strings.
- ➜ Better than in the paper, also removes categories with only infinite strings.
- ➜ Example causes:
  - ➜ Agreement with features not in the lexicon.
  - ➜ Cycle elimination.

Rules removed from the French example:

```
NP.s → Spec.n=Pl;.s!Fem!Nom Noun.g=Fem;.s!Pl
NP.s → Spec.n=Sg;.s!Fem!Nom Noun.g=Fem;.s!Sg
Noun.g=Fem;.s!Pl → Noun.g=Fem;.s!Pl "noires"
Noun.g=Fem;.s!Sg → Noun.g=Fem;.s!Sg "noire"
```

TOP-DOWN FILTERING

- Removes productions for unreachable categories.
- Example causes:
  - Choice of start category, e.g. in multimodal grammars.
  - Unused forms.
  - Cycle elimination.
  - Bottom-up filtering.

Categories removed from the English example:

`Noun{}.s!Sg!Acc`, `Noun{}.s!Sg!Gen`,
`Noun{}.s!Pl!Acc`, `Noun{}.s!Pl!Gen`.

LEFT RECURSION ELIMINATION

➜ CLC_{LR} transform by Moore (2000).

Left-recursive rules in the French example:

```
Noun.g=Masc;.s!Sg → Noun.g=Masc;.s!Sg "noir"
Noun.g=Masc;.s!Pl → Noun.g=Masc;.s!Pl "noirs"
```

After LC_{LR} transform (only singular shown):

```
Noun.g=Masc;.s!Sg → "chat" Noun.g=Masc;.s!Sg-"chat"
Noun.g=Masc;.s!Sg-"chat" →
Noun.g=Masc;.s!Sg-"chat" →
            Noun.g=Masc;.s!Sg-Noun.g=Masc;.s!Sg
Noun.g=Masc;.s!Sg-Noun.g=Masc;.s!Sg → "noir"
Noun.g=Masc;.s!Sg-Noun.g=Masc;.s!Sg →
          "noir" Noun.g=Masc;.s!Sg-Noun.g=Masc;.s!Sg
```

IDENTICAL CATEGORY ELIMINATION

- ➜ Merges categories with the same set of right-hand sides.
- ➜ Causes:
  - ➜ Words with multiple identical forms.

Merged rules in the Swedish example:

```
Spec.det=DIndef;.n=Pl;.s!False!Utr --> "två"
Spec.det=DIndef;.n=Pl;.s!True!Utr --> "två"
Spec.det=DIndef;.n=Sg;.s!False!Utr --> "en"
Spec.det=DIndef;.n=Sg;.s!True!Utr --> "en"
```

CFG SIZE REDUCTION

Language	Initial	Bottom-up	Top-down	LC_{LR}	Merge
English	22	22	10	10	10
French	28	24	10	20	20
Spanish	28	24	10	20	20
Italian	44	40	10	20	20
Swedish	116	72	18	18	16
Danish	116	72	18	18	16
German	118	94	16	16	15
Finnish	153	143	14	14	14
Norwegian	156	76	18	18	16
Russian	364	234	12	12	12

EBNF COMPACTION

- Makes regular expressions from terminal sequences.
- Example: 10x GSL size reduction of Perera's and Ranta's SAMMIE grammar.
- Causes:
  - Use of **variants**.

OUTPUT FORMAT: NUANCE GSL

```
> pg -printer=gsl

;GSL2.0
.MAIN Toy0Eng_0
Toy0Eng_0 [(Toy0Eng_3 Toy0Eng_1)
           (Toy0Eng_4 Toy0Eng_2)]
Toy0Eng_1 [("black" Toy0Eng_1) "dogs" "cats"]
Toy0Eng_2 [("black" Toy0Eng_2) "dog" "cat"]
Toy0Eng_3 "two"
Toy0Eng_4 "one"
```

OUTPUT FORMAT: SRGS (XML)

```
> pg -printer=srgs_xml
```

```
...
  <rule id="Toy0Dan_1">
    <one-of>
      <item>hunder</item>
      <item>katte</item>
    </one-of>
  </rule>
  <rule id="Toy0Dan_3">
    <item>
      <item>sorte</item>
      <ruleref uri="#Toy0Dan_5" />
    </item>
  </rule>
...
  <rule id="Toy0Dan_5">
    <one-of>
      <item><ruleref uri="#Toy0Dan_1" /></item>
      <item><ruleref uri="#Toy0Dan_3" /></item>
    </one-of>
  </rule>
...
```

OUTPUT FORMAT: SRGS (ABNF)

```
> pg -printer=srgs_abnf

#ABNF 1.0 UTF-8;
language en-US;
root $NP_cat;
public $NP_cat = $Toy0Eng_0 ;
public $Noun_cat = $Toy0Eng_1 | $Toy0Eng_2 ;
public $Spec_cat = $Toy0Eng_3 | $Toy0Eng_4 ;
$Toy0Eng_0 = $Toy0Eng_3 $Toy0Eng_1
             | $Toy0Eng_4 $Toy0Eng_2 ;
$Toy0Eng_1 = black $Toy0Eng_1 | dogs | cats ;
$Toy0Eng_2 = black $Toy0Eng_2 | dog | cat ;
$Toy0Eng_3 = two ;
$Toy0Eng_4 = one ;
```

OUTPUT FORMAT: JSGF

```
> pg -printer=jsgf

#JSGF V1.0 UTF-8 en-US;
grammar Toy0Eng;
public <MAIN> = <NP_cat> ;
public <NP_cat> = <Toy0Eng_0> ;
public <Noun_cat> = <Toy0Eng_1> | <Toy0Eng_2> ;
public <Spec_cat> = <Toy0Eng_3> | <Toy0Eng_4> ;
<Toy0Eng_0> = <Toy0Eng_3> <Toy0Eng_1>
              | <Toy0Eng_4> <Toy0Eng_2> ;
<Toy0Eng_1> = black <Toy0Eng_1> | dogs | cats ;
<Toy0Eng_2> = black <Toy0Eng_2> | dog | cat ;
<Toy0Eng_3> = two ;
<Toy0Eng_4> = one ;
```

FINITE AUTOMATA GENERATION

① Regular approximation:
- ➜ Approximation: context-free $\Rightarrow$ regular.
- ➜ Algorithm by Mohri and Nederhof (2001).

② Finite automata compilation:
- ➜ Regular grammar $\Rightarrow$ labelled NFAs.
- ➜ One automaton per category.
- ➜ Non-mutually recursive categories treated as terminals.
- ➜ Modified version of algorithm by Nederhof (2000).

③ Finite automata minimization:
- ➜ NFAs $\Rightarrow$ minimal DFAs.
- ➜ Algorithm by Brzozowski (1962).

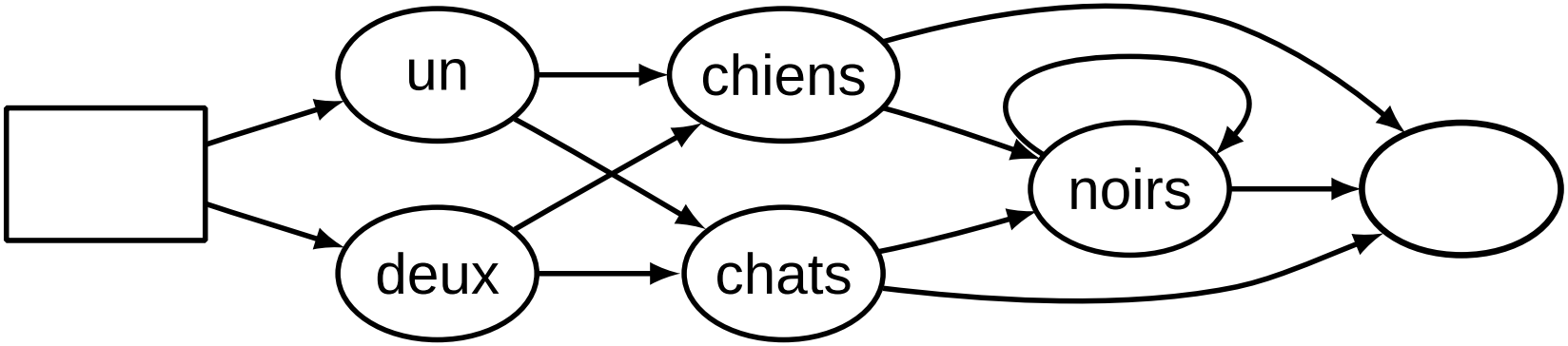
OUTPUT FORMAT: SLF

```
> pg -printer=slf
```

```
N=10 L=18
I=0 W=!NULL
I=1 W=!NULL
I=2 W=DOG s=dog
I=3 W=CAT s=cat
I=4 W=DOGS s=dogs
I=5 W=CATS s=cats
I=6 W=BLACK s=black
I=7 W=TWO s=two
I=8 W=BLACK s=black
I=9 W=ONE s=one
J=0 S=2 E=0
J=1 S=3 E=0
J=2 S=4 E=0
J=3 S=5 E=0
J=4 S=6 E=5
J=5 S=6 E=4
J=6 S=7 E=5
J=7 S=7 E=4
J=8 S=6 E=6
J=9 S=7 E=6
J=10 S=1 E=7
J=11 S=8 E=3
J=12 S=8 E=2
J=13 S=9 E=3
J=14 S=9 E=2
J=15 S=8 E=8
J=16 S=9 E=8
J=17 S=1 E=9
```

Output format: SLF (Graphviz)

```
> pg -printer=slf_graphviz
```



OUTPUT FORMAT: REGULAR EXPRESSION

```
> pg -printer=regexp

en sort* (hund | kat) | to sorte* (hunder | katte)

en svart* (hund | katt) | två svarta* (hundar | katter)

en svart* (hund | katt) | to svarte* (hunder | katter)

ein schwarzer* Hund | eine schwarze* Katze

                    | zwei schwarze* (Hunde | Katzen)

one black* (cat | dog) | two black* (cats | dogs)

dos (gatos | perros) negros* | uno (gato | perro) negro*

deux (chats | chiens) noirs* | un (chat | chien) noir*

due (cani | gatti) neri* | uno (cane | gatto) nero*

kaksi mustaa* (kissaa | koiraa) | yksi musta* (kissa | koira)
```

NON-RECURSIVE GRAMMARS

- New development, not in the paper.
- Nuance Recognizer 9.0 uses SRGS (XML), without recursion.
- Algorithm:
  1. Generate a set of labeled finite automata.
  2. Convert each automaton to a regular expression.
  3. Output each labeled regular expression as an EBNF production.

NON-RECURSIVE SRGS (ABNF)

```
> pg -printer=srgs_abnf_non_rec

#ABNF 1.0 UTF-8;
language en-US;
root $NP_cat;
public $NP_cat = $Toy0Eng_4 ;
public $Noun_cat = $Toy0Eng_2 | $Toy0Eng_3 ;
public $Spec_cat = $Toy0Eng_0 | $Toy0Eng_1 ;
$Toy0Eng_0 = one ;
$Toy0Eng_1 = two ;
$Toy0Eng_2 = black black<0-> (cat | dog) | cat
             | dog ;
$Toy0Eng_3 = black black<0-> (cats | dogs) | cats
             | dogs ;
$Toy0Eng_4 = $Toy0Eng_1 $Toy0Eng_3
             | $Toy0Eng_0 $Toy0Eng_2 ;
```

SEMANTICS

- Semantic Interpretation for Speech Recognition (SISR)
- Embedded ECMAScript code.
- Builds abstract syntax trees.
- Carried through left-recursion elimination by using λ terms.
- Similar to algorithm by Bos (2002).

JSGF + SISR

```
> pg -printer=jsgf_sisr_old

...
<Toy0Eng_0> =
  <NULL> { var a = [] }
  (<Toy0Eng_4> { a[0] = $Toy0Eng_4 }
   <Toy0Eng_2> { a[1] = $Toy0Eng_2 })
  { $ = { name: "SpecNoun", args: [a[0], a[1]] \} };
<Toy0Eng_2> =
  <NULL> { var a = [] }
  (black <Toy0Eng_2> { a[0] = $Toy0Eng_2 })
                { $ = { name: "Black", args:[a[0]] \} }
 | (dog) { $ = { name: "Canis", args: [] \} }
 | (cat) { $ = { name: "Felis", args: [] \} };
<Toy0Eng_4> = (one)
                { $ = { name: "One", args: [] \} };
...
```

CONCLUSIONS

- Compilation: GF $\Rightarrow$ speech recognition grammar.
- Can use GF resource grammars.
- Many formats, both context-free and regular.
- Generates compact grammars.
- Can include semantics.