Detecting Ambiguity in Programming Language Grammars

Naveneetha Krishnan Vasudevan

Laurence Tratt

KING'S College LONDON

Software Development Team

2013-10-27

Overview

Non-deterministic ambiguity detector

Simpler

Yet performs 11% better

Other Contributions

- New approaches to generating grammars

Other Contributions

- New approaches to generating grammars
- Breadth-over-depth non-deterministic approach outperforms more complex heuristics

I. Ambiguity

What is Ambiguity?

- Undecidable problem
- Undesirable

Multiple Parses

```
E -> E '+' E  |  E '*' E
```

Multiple Parses

```
E -> E '+' E  |  E '*' E
```

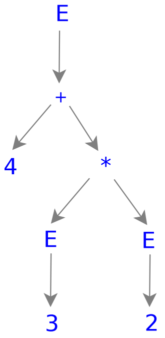
```
4 + 3 * 2
```

Multiple Parses

```
E -> E '+' E  |  E '*' E
```

4 + 3 * 2

Tree 1

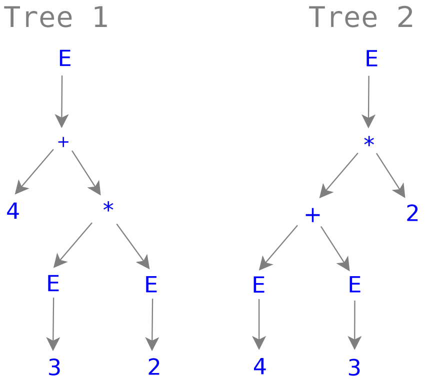


value = 10

Multiple Parses

```
E -> E '+' E  |  E '*' E
```

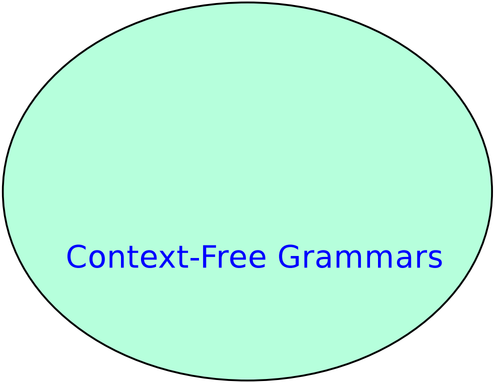
$4 + 3 * 2$



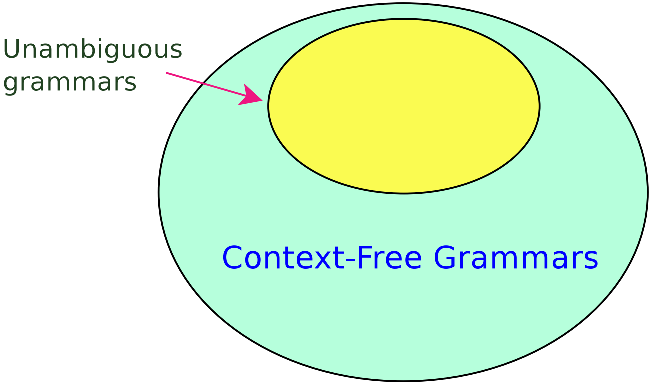
value = 10

value = 14

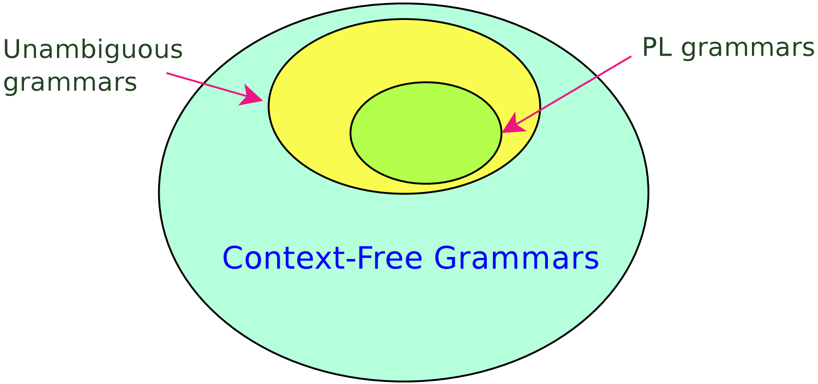
CFGs



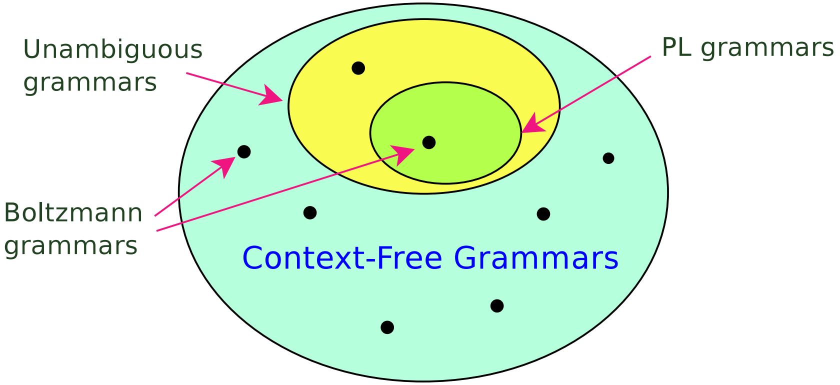
CFGs



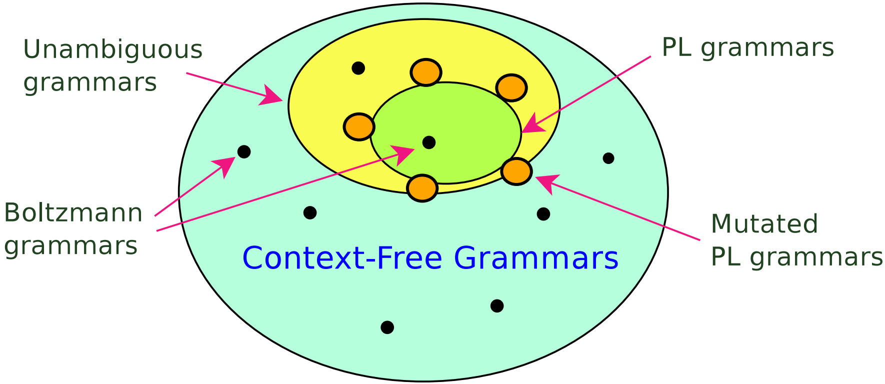
CFGs



CFGs


Unambiguous grammars
PL grammars
Boltzmann grammars
Context-Free Grammars

CFGs


Unambiguous grammars
PL grammars
Boltzmann grammars
Mutated PL grammars
Context-Free Grammars

SinBAD

$\triangleq$

Search Based Ambiguity Detection

Search-Based

=

aim to find **good enough** solutions

Pure Random

Pure random

=

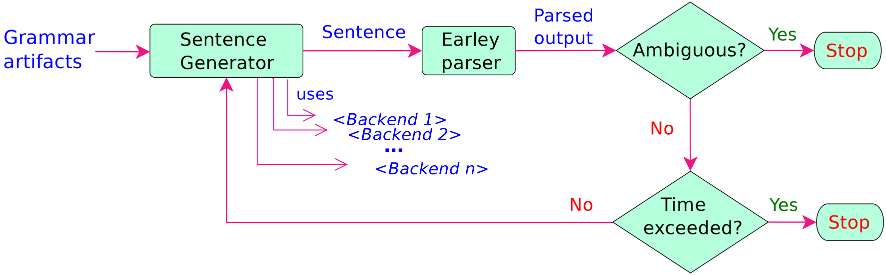
simple + scan randomly

Heuristics

=

fitness function + guided search

SinBAD



Grammar artifacts
Sentence Generator
Sentence
Earley parser
Parsed output
Ambiguous?
Yes
Stop
No
uses
<Backend 1>
<Backend 2>
...
<Backend n>
No
Time exceeded?
Yes
Stop

dynamic1

$$\triangleq$$

non-deterministic approach
to ambiguity detection

dynamic1

generate(G, *rule*, d, D, *sentence*):

```
if (d > D)
    alt ← favour-alternative(rule)
else
    alt ← pick an alternative randomly
for α ∈ alt
    if α ∈ non-terminals
        alt → generate(..,d+1,..)
    else
        sentence.append(α)
Keep count of rule.entered and rule.exited
```

dynamic1

generate(G, *rule*, d, D, *sentence*):
    if ($d > D$)
        alt $\leftarrow$ favour-alternative(*rule*)
    else
        alt $\leftarrow$ pick an alternative randomly
    for $\alpha \in$ *alt*
        if $\alpha \in$ non-terminals
            alt $\rightarrow$ generate(..,d+1,..)
        else
            sentence.append(α)
    Keep count of *rule.entered* and *rule.exited*

dynamic1

```
generate(G, rule, d, D, sentence):
    if (d > D)
        alt ← favour-alternative(rule)
    else
        alt ← pick an alternative randomly
    for α ∈ alt
        if α ∈ non-terminals
            alt → generate(..,d+1,..)
        else
            sentence.append(α)
    Keep count of rule.entered and rule.exited
```

dynamic1

generate(G, *rule*, d, D, *sentence*):
if ($d > D$)
alt $\leftarrow$ favour-alternative(*rule*)
else
alt $\leftarrow$ pick an alternative randomly
for $\alpha \in$ *alt*
if $\alpha \in$ non-terminals
alt $\rightarrow$ generate(..,d+1,..)
else
sentence.append(α)
Keep count of *rule.entered* and *rule.exited*

Favour Alternative

favour-alternative(*rule*):
    scores $\leftarrow$ [ ]
    for *alt* $\in$ *rule*
        score $\leftarrow$ 0
        for $\alpha \in$ *alt*
            if $\alpha \in$ non-terminals
                score += (1 - (α.*exited*/α.*entered*))
        scores $\leftarrow$ *score*
    alts $\leftarrow$ { alternatives of *rule* whose score = min(*scores*) }
    return random(*alts*)

Favour Alternative

favour-alternative(*rule*):
    scores $\leftarrow$ [ ]
    for *alt* $\in$ *rule*
        score $\leftarrow$ 0
        for $\alpha \in$ *alt*
            if $\alpha \in$ non-terminals
                score += (1 - (α.*exited*/α.*entered*))
        scores $\leftarrow$ *score*
    alts $\leftarrow$ { alternatives of *rule* whose score = min(*scores*) }
    return random(*alts*)

Favour Alternative

favour-alternative(*rule*):
  scores $\leftarrow$ [ ]
  for *alt* $\in$ *rule*
    score $\leftarrow$ 0
    for $\alpha \in$ *alt*
      if $\alpha \in$ non-terminals
        score += (1 - (α.*exited*/α.*entered*))
    scores $\leftarrow$ *score*
  alts $\leftarrow$ { alternatives of *rule* whose score = min(*scores*) }
  return random(*alts*)

Favour Alternative

```
favour-alternative(rule):
    scores ← [ ]
    for alt ∈ rule
        score ← 0
        for α ∈ alt
            if α ∈ non-terminals
                score += (1 - (α.exited/α.entered))
        scores ← score
    alts ← { alternatives of rule whose score = min(scores) }
    return random(alts)
```

II. Grammar Generation

Random grammars

$\triangleq$

based on Boltzmann sampling

Boltzmann Specification for CFGs

Cfg = Cfg Rule ... Rule

Rule = SingleAlt Alt | RuleAlts1 Rule Alt

Alt = EmptyAltSyms | SingleAltSyms1 Symbol | AltSyms1 Alt Symbol

Symbol = NonTerm NonTerm | Term Term

NonTerm = NonTerm1 | NonTerm2 | ... | NonTermN

Term = Term1 | Term2 | ... | TermN

Boltzmann Specification for CFGs

Cfg = Cfg Rule ... Rule

Rule = SingleAlt Alt | RuleAlts1 Rule Alt

Alt = EmptyAltSyms | SingleAltSyms1 Symbol | AltSyms1 Alt Symbol

Symbol = NonTerm NonTerm | Term Term

NonTerm = NonTerm1 | NonTerm2 | ... | NonTermN

Term = Term1 | Term2 | ... | TermN

Boltzmann Specification for CFGs

Cfg = Cfg Rule ... Rule

Rule = SingleAlt Alt | RuleAlts1 Rule Alt

Alt = EmptyAltSyms | SingleAltSyms1 Symbol | AltSyms1 Alt Symbol

Symbol = NonTerm NonTerm | Term Term

NonTerm = NonTerm1 | NonTerm2 | ... | NonTermN

Term = Term1 | Term2 | ... | TermN

Boltzmann Specification for CFGs

Cfg = Cfg Rule ... Rule

Rule = SingleAlt Alt | RuleAlts1 Rule Alt

Alt = EmptyAltSyms | SingleAltSyms1 Symbol | AltSyms1 Alt Symbol

Symbol = NonTerm NonTerm | Term Term

NonTerm = NonTerm1 | NonTerm2 | ... | NonTermN

Term = Term1 | Term2 | ... | TermN

Mutated grammar

=

unambiguous grammar

+

single mutation

Types of Mutations

- Add empty
- Mutate symbol
- Add symbol
- Delete symbol

Ambiguity is statically undecidable: so how do we evaluate a tool?

III. Experiment

Experiment Methodology

- Comparative study
- Grammar sets
- Sub-experiments: mini, main and verification

Tools Selected

Tools:

- ACLA (approximation)
- AMBER (exhaustive search)
- AmbiDexter (filtering + search)
- dynamic1 (non-deterministic)

Various tools and options

$+$

lots of grammars

$\triangleq$

mini experiment

Mini Experiment

- 400 random grammars

Mini Experiment

- 400 random grammars
- 160 mutated PL grammars

Mini Experiment

- 400 random grammars
- 160 mutated PL grammars
- 20 manually modified PL grammars

Mini Experiment

- 400 random grammars
- 160 mutated PL grammars
- 20 manually modified PL grammars
- Time limits: 10s, 30s, 60s, and 120s

Best options → main experiment

Mini Experiment

- 400 random grammars
- 160 mutated PL grammars
- 20 manually modified PL grammars
- Time limits: 10s, 30s, 60s, and 120s

Best options $\rightarrow$ main experiment

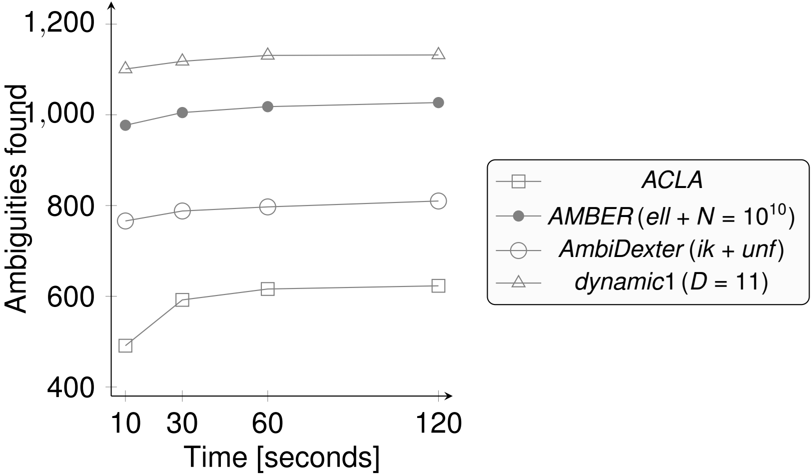
Main experiment

=

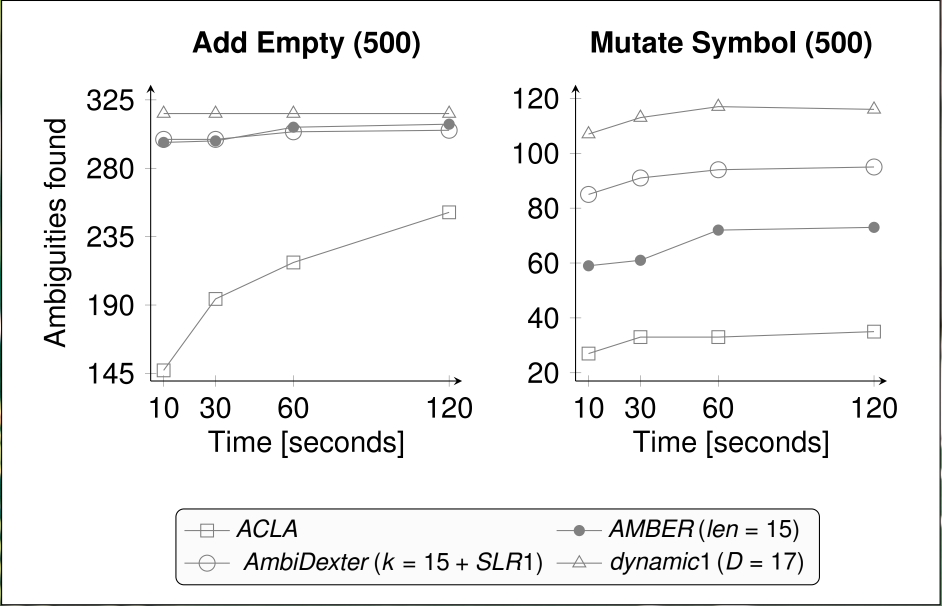
mini experiment $\times$ 7

Results

Boltzmann sampled grammars (1600)



Results


Add Empty (500)
Mutate Symbol (500)
Ambiguities found
325
280
235
190
145
120
100
80
60
40
20
10
30
60
120
Time [seconds]
ACLA
AMBER (len = 15)
AmbiDexter (k = 15 + SLR1)
dynamic1 (D = 17)

Verification experiment

$$=$$

main experiment $\times$ 5

Summary

dynamic1:

- Simple heursitic

Summary

dynamic1:

- Simple heursitic
- Performs as well as other tools

Summary

dynamic1:

- Simple heursitic
- Performs as well as other tools
- Does well on long ambiguous subsets

dynamic1:

- Simple heursitic
- Performs as well as other tools
- Does well on long ambiguous subsets
- Breadth-over-depth performs better

Summary

dynamic1:

- Simple heursitic
- Performs as well as other tools
- Does well on long ambiguous subsets
- Breadth-over-depth performs better

http://soft-dev.org/