Automated Ambiguity Detection in Layout-Sensitive Grammars

Jiangyi LIU[†*][1] Fengmin (Paul) ZHU[†][2] Fei HE[3]

October 27, 2023

[1]Tsinghua University and University of Wisconsin-Madison

[2]Tsinghua University and CISPA Helmholtz Center for Information Security

[3]Tsinghua University

[†] Equally contributed.

[*] This work was done at Tsinghua University.

What Are Layout-Sensitive Grammars?

Grammars where indentations and whitespaces affect parsing.

```python
if False:
    print(1, end=' ')
print(2)
# Output: 2
```

```python
if False:
    print(1, end=' ')
    print(2)
# Output: Nothing
```

What Are Layout-Sensitive Grammars?

Grammars where indentations and whitespaces affect parsing.

```
if False:
    print(1, end=' ')
print(2)
# Output: 2
```

```
if False:
    print(1, end=' ')
    print(2)
# Output: Nothing
```

Why Using Them?

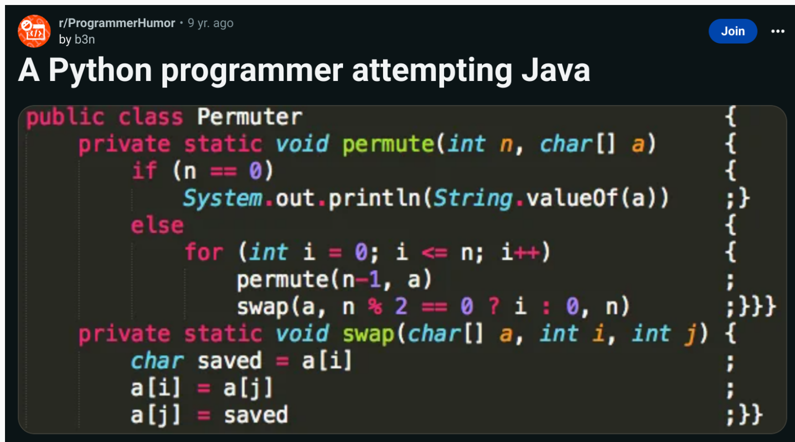
- Elegant and stylized; no disturbing symbols

Why Using Them?

- Elegant and stylized; no disturbing symbols
- Readability counts — Zen of Python

Why Using Them?

- Elegant and stylized; no disturbing symbols
- Readability counts — Zen of Python



Why Using Them?

- Elegant and stylized; no disturbing symbols
- Readability counts — Zen of Python

That programmer would love Scala 3!

```
object Permuter:
  private def permute(n: Int, a: Array[Char]): Unit =
    if n == 0 then
      println(String.valueOf(a))
    else
      for i <- 0 to n do
        permute(n-1, a)
        swap(a, if n % 2 == 0 then i else 0, n)
  private def swap(a: Array[Char], i: Int, j: Int) =
    val saved = a(i)
    a(i) = a(j)
    a(j) = saved
```

Ambiguity Matters

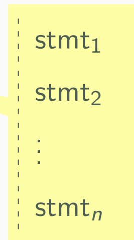
A useful new layout-sensitive grammar should be unambiguous.

Manually Checking Ambiguity is Hard

Let's consider a layout-sensitive grammar:

$$\text{block} \rightarrow \|\text{stmt}\|^{+}$$

$$\text{stmt} \rightarrow \textbf{nop} \mid \textbf{do}\ \text{block}$$

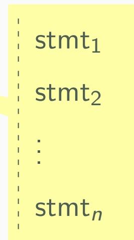


Manually Checking Ambiguity is Hard

Let's consider a layout-sensitive grammar:

$$\text{block} \rightarrow \|\text{stmt}\|^{+}$$

$$\text{stmt} \rightarrow \textbf{nop} \mid \textbf{do}\ \text{block}$$



Q: Is this grammar ambiguous?

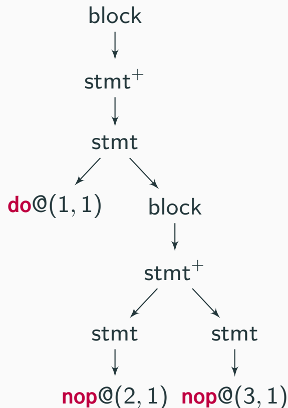
It Is Ambiguous

do

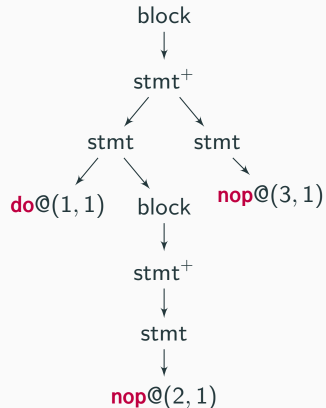
nop

nop

[**do**@(1, 1),
nop@(2, 1),
nop@(3, 1)]



parse tree t_1



parse tree t_2

Lessons

Ambiguity should be detected automatically, not manually.

Lessons

Ambiguity should be detected automatically, not manually.

- Q: But isn't ambiguity undecidable?

Lessons

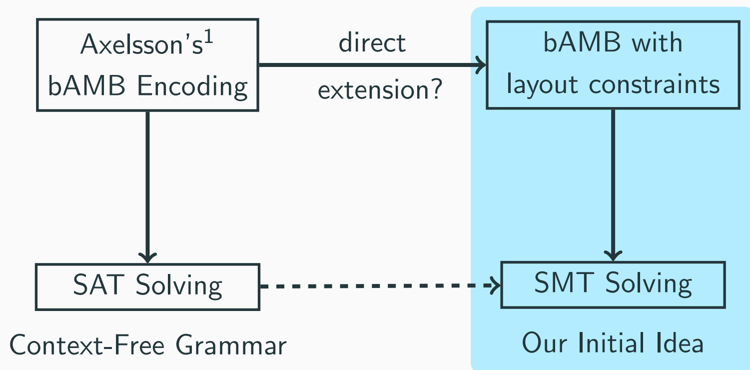
Ambiguity should be detected automatically, not manually.

- Q: But isn't ambiguity undecidable?
- A: Find an incomplete but useful way.

Bounded Ambiguity Checking

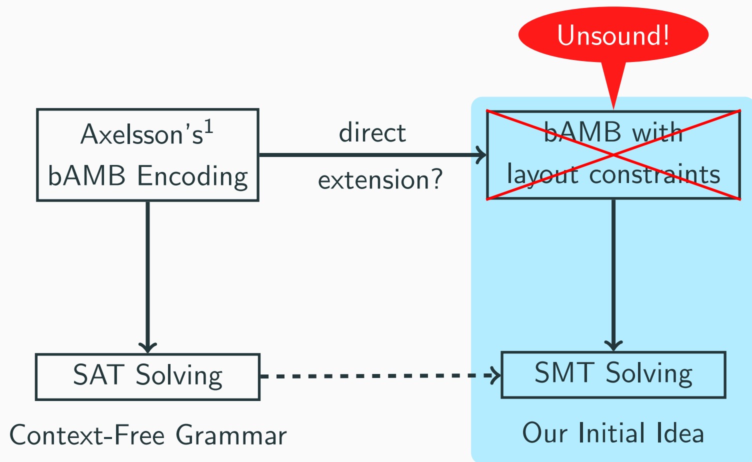
Definition (Bounded ambiguity)
Given a layout-sensitive grammar G, is there a sentence of length up to k that has multiple parse trees?

Bounded Ambiguity Checking



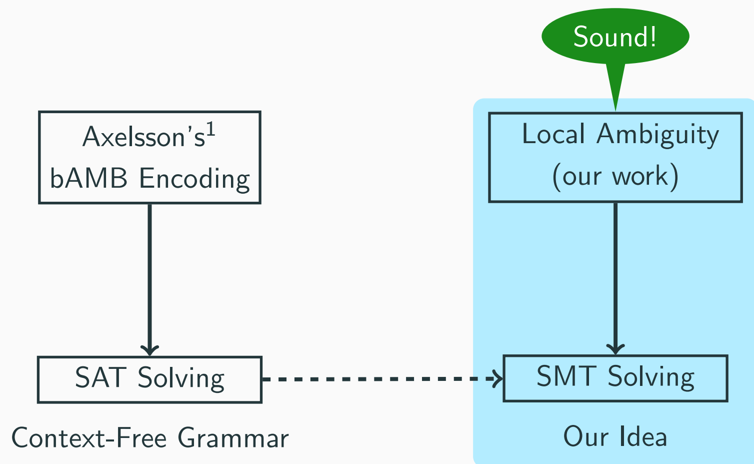
[1] Axelsson, Roland, Keijo Heljanko, and Martin Lange. "Analyzing context-free grammars using an incremental SAT solver." ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35. Springer Berlin Heidelberg, 2008.

Bounded Ambiguity Checking



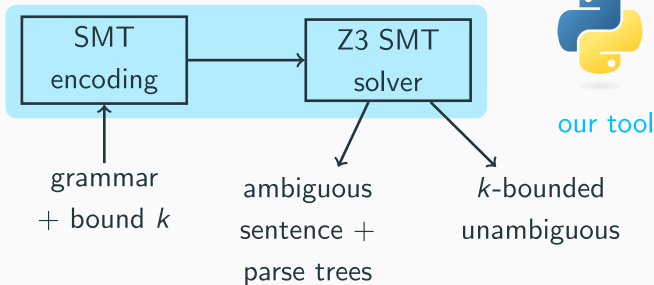
[1] Axelsson, Roland, Keijo Heljanko, and Martin Lange. "Analyzing context-free grammars using an incremental SAT solver." ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35. Springer Berlin Heidelberg, 2008.

Bounded Ambiguity Checking

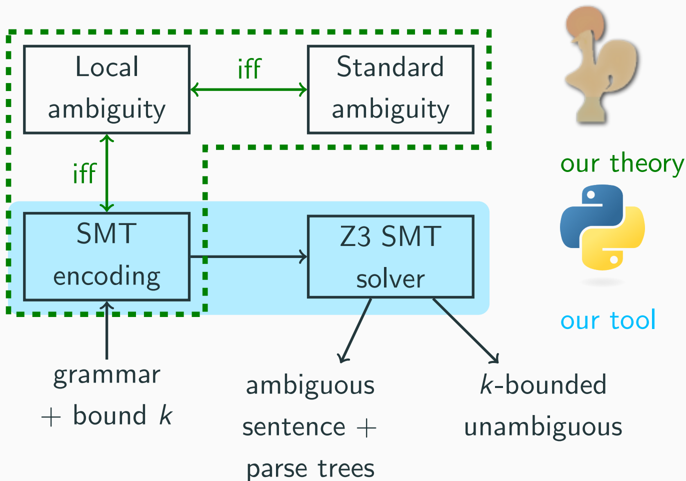


[1] Axelsson, Roland, Keijo Heljanko, and Martin Lange. "Analyzing context-free grammars using an incremental SAT solver." ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35. Springer Berlin Heidelberg, 2008.

Framework for Layout-Sensitive Ambiguity Detection



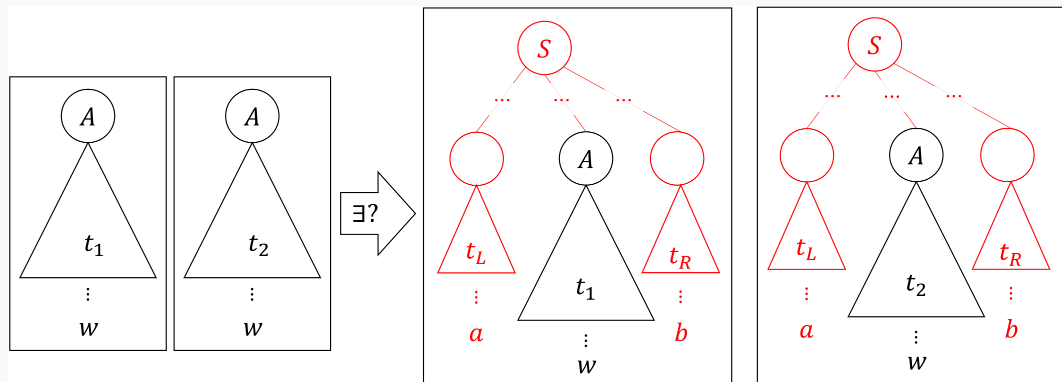
Framework for Layout-Sensitive Ambiguity Detection



our theory

our tool

Axelsson's bAMB Encoding



bAMB = subtrees differ on the first level

Axelsson's bAMB is Unsound

The following grammer is unambiguous:

$$S \to (\!| A |\!) \qquad A \to B_1 \qquad A \to B_2$$

$$B_1 \to C \parallel C \qquad B_2 \to C\ C \qquad C \to c$$

Axelsson's bAMB is Unsound

The following grammer is unambiguous:

$$S \to (\!|A|\!) \qquad A \to B_1 \qquad A \to B_2$$

$$B_1 \to C \parallel C \qquad B_2 \to C\ C \qquad C \to c$$

$(\!|\cdot|\!)$: all tokens derived are on the same line

Axelsson's bAMB is Unsound

The following grammer is unambiguous:

$$S \to (\!|A|\!) \qquad A \to B_1 \qquad A \to B_2$$

$$B_1 \to C \parallel C \qquad B_2 \to C\ C \qquad C \to c$$

$\cdot \parallel \cdot$: first tokens produced by both sides lie at the same column

Axelsson's bAMB is Unsound

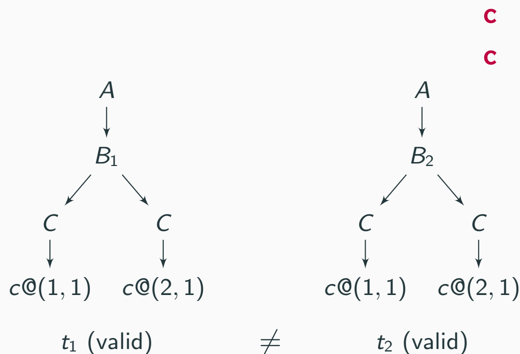
$$S \rightarrow (\!|A|\!) \qquad A \rightarrow B_1 \qquad A \rightarrow B_2$$

$$B_1 \rightarrow C \parallel C \qquad B_2 \rightarrow C\ C \qquad C \rightarrow c$$

... but the following example satisfies bAMB:

c

c

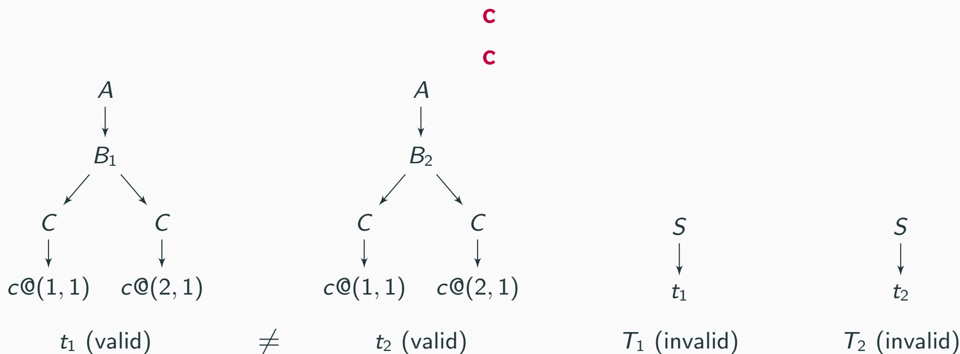


Axelsson's bAMB is Unsound

$$S \to (\!|A|\!) \qquad A \to B_1 \qquad A \to B_2$$

$$B_1 \to C \parallel C \qquad B_2 \to C\ C \qquad C \to c$$

... but the following example satisfies bAMB:



Why This Happens to Layout-Sensitive Grammars?

The complete parse trees can still be malformed/invalid,
even if valid ambiguous subtrees exist!

Why This Happens to Layout-Sensitive Grammars?

The complete parse trees can still be malformed/invalid,
even if valid ambiguous subtrees exist!

Solution: drop subtrees that are unreachable from the start symbol.

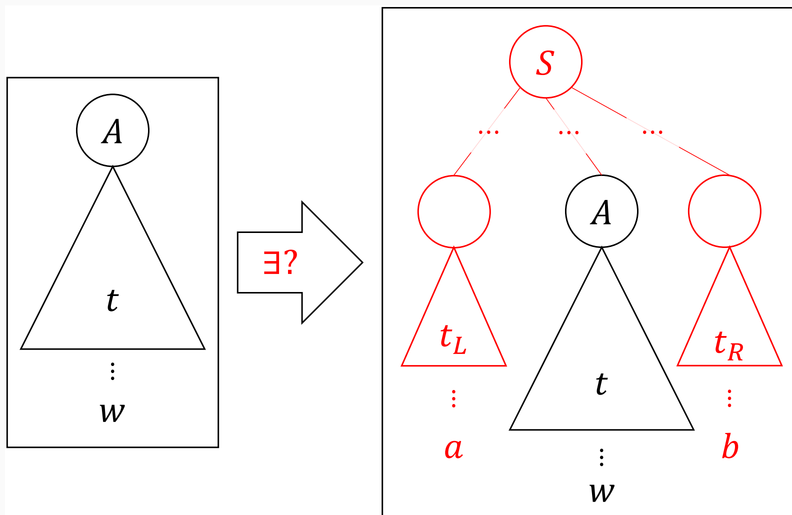
Reachability (for Context-Free Grammars)

Definition
Nonterminal A is reachable[2] in context-free grammar G from symbol S if there exists a derivation $S \Rightarrow_* \alpha A \beta$ for some sentential forms α and β.

[2] Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2006. "Introduction to automata theory, languages, and computation (3rd Edition)." Addison-Wesley Longman Publishing Co., Inc., USA.

Reachability (for Layout-Sensitive Grammars)

Given a subtree with A as root and derives sentence w



Local Ambiguity

Intuitively, local ambiguity is essentially

$$\text{bAMB} \land \text{reachability}$$

Local Ambiguity

Intuitively, local ambiguity is essentially

$$\text{bAMB} \wedge \text{reachability}$$

Theorem
(Equivalence between (standard) ambiguity and local ambiguity)
(A, w) *is locally ambiguous iff the derivation* $A \Rightarrow_* w$ *is ambiguous.*

Encoding Local Ambiguity

$$\Phi(k) \triangleq \Phi_D(k) \wedge \Phi_R^{\varepsilon}(k) \wedge \Phi_R^{\not\varepsilon}(k)$$
$$\wedge \bigvee_{H \in N} \left((\mathcal{R}_{\varepsilon}^{H} \wedge \Phi_{\text{multi}}(H,0,0)) \vee \bigvee_{\substack{0<\delta \\ x+\delta \leq k}} (\mathcal{R}_{x,\delta}^{H} \wedge \Phi_{\text{multi}}(H,x,\delta)) \right)$$

- $\Phi_D(k)$: the sentence can be derived in given grammar
- $\Phi_{\text{multi}}(H,x,\delta)$: at least two parse trees exist

} bAMB

- $\Phi_R^{\varepsilon}(k)$, $\Phi_R^{\not\varepsilon}(k)$, $\mathcal{R}_{\varepsilon}^{H}$, $\mathcal{R}_{x,\delta}^{H}$: reachability conditions

Soundness & Completeness of Encoding

Our encoding $\Phi(k)$ is bounded sound and complete:

Soundness Any sentence w that satisfies $\Phi(k) \Rightarrow w$ is ambiguous.

Completeness If there exists an ambiguous sentence of length k, then $\Phi(k)$ is satisfiable.

Take-Home Messages

- Ambiguity is important in language design.
- Proof assistant can help with finding design flaws in encoding.
- SMT solving is powerful for formal language research.

Automated Ambiguity Detection in Layout-Sensitive Grammars

JIANGYI LIU*, Tsinghua University, China
FENGMIN ZHU*, Tsinghua University, China and CISPA Helmholtz Center for Information Security, Germany
FEI HE†, Tsinghua University, China, Key Laboratory for Information System Security, Ministry of Education, China, and Beijing National Research Center for Information Science and Technology, China

Take-Home Messages

- Ambiguity is important in language design.
- Proof assistant can help with finding design flaws in encoding.
- SMT solving is powerful for formal language research.

Check our website:

Automated Ambi … out-Sensitive Grammars

JIANGYI LIU*, Tsinghua Ur
FENGMIN ZHU*, Tsinghu … oltz Center for Information Security, Germany
FEI HE†, Tsinghua University … System Security, Ministry of Education, China, and Beijing National R … and Technology, China