Automated Ambiguity Detection in Layout-Sensitive Grammars

Jiangyi LIU[1] Fengmin (Paul) ZHU[1,2] Fei HE[1]

[1]School of Software, Tsinghua University [2]CISPA Helmholtz Center for Information Security

Layout-Sensitive Languages

Whitespaces and indentations affect how programs get parsed.

```
def greeting():
  print('Welcome to our poster' +
    ' and our talk at 15:12 Friday!')
```

$\text{indent}(w_1, w_2) \triangleq (w_1 \neq \varepsilon \wedge w_2 \neq \varepsilon)$
$\Rightarrow w_2[0].\text{col} > w_1[0].\text{col} \wedge w_2[0].\text{line} = w_1[-1].\text{line} + 1$

```
match solver_result with
| Sat model -> Ambig (decode model)
| Unsat -> Unambig
| _ -> Unknown
```

$\text{align}(w_1, w_2) \triangleq (w_1 \neq \varepsilon \wedge w_2 \neq \varepsilon) \Rightarrow w_1[0].\text{col} = w_2[0].\text{col}$

```
do grammar <- load "my_lang.ebnf"
   sentence <- checkAmbig grammar
   return $ AmbigResult sentence $
     trees grammar sentence
```

$\text{offside}(w) \triangleq w \neq \varepsilon$
$\Rightarrow \forall t \in w : t.\text{line} > w[0].\text{line} \Rightarrow t.\text{col} > w[0].\text{col}$

Ambiguity Matters

Consider a grammar fragment:

$$\text{block} \to \text{stmt}^*$$
$$\text{stmt} \to \texttt{var = expr} \mid \texttt{while expr do block} \mid \cdots$$

This sentence has two different parses that are semantically different:

```
while x > 0 do
x = x - 1
x = x + 1
```

parse 1

```
while (x > 0) {
  x = x - 1;
  x = x + 1;
}
```

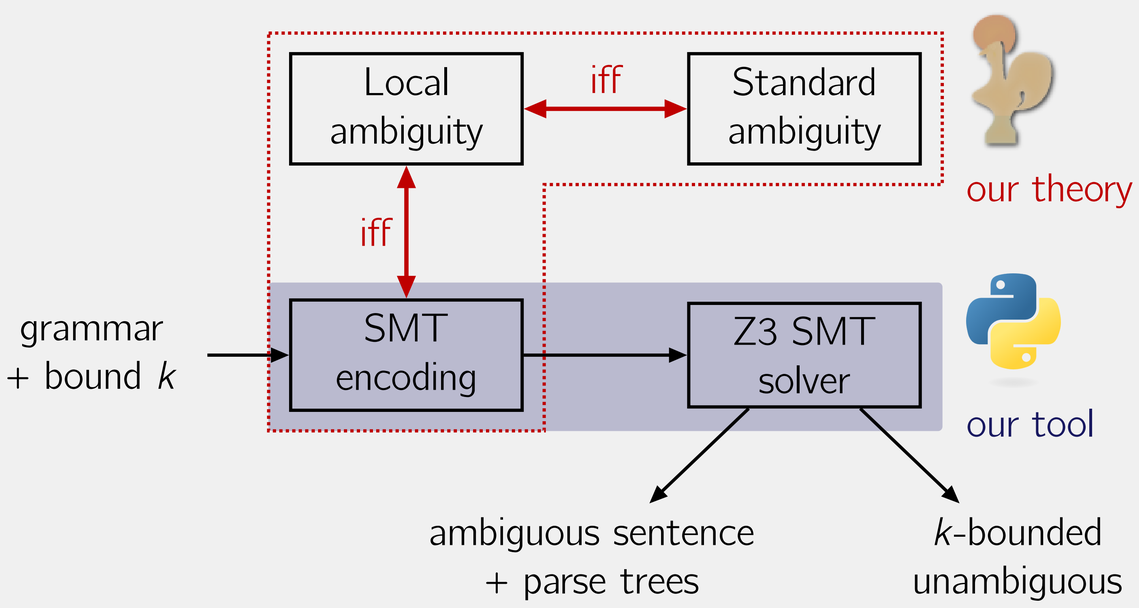
nonterminating if $x > 0$

parse 2

```
while (x > 0) {
  x = x - 1;
}
x = x + 1;
```

terminating

Lamb: Layout-Sensitive Ambiguity Detector



where k is the upper bound length of the sentences being considered

A Tour of Lamb

Step 1: Input a grammar G_{block}:

$$\text{block} \to \|\text{stmt}\|^+$$
$$\text{stmt} \to \text{nop} \mid \text{do block}$$

The alignment constraint $\|\cdot\|^+$ marks the border of the do-block body, so it distinguishes between

```
do nop
   nop
```

and

```
do nop
   nop
```

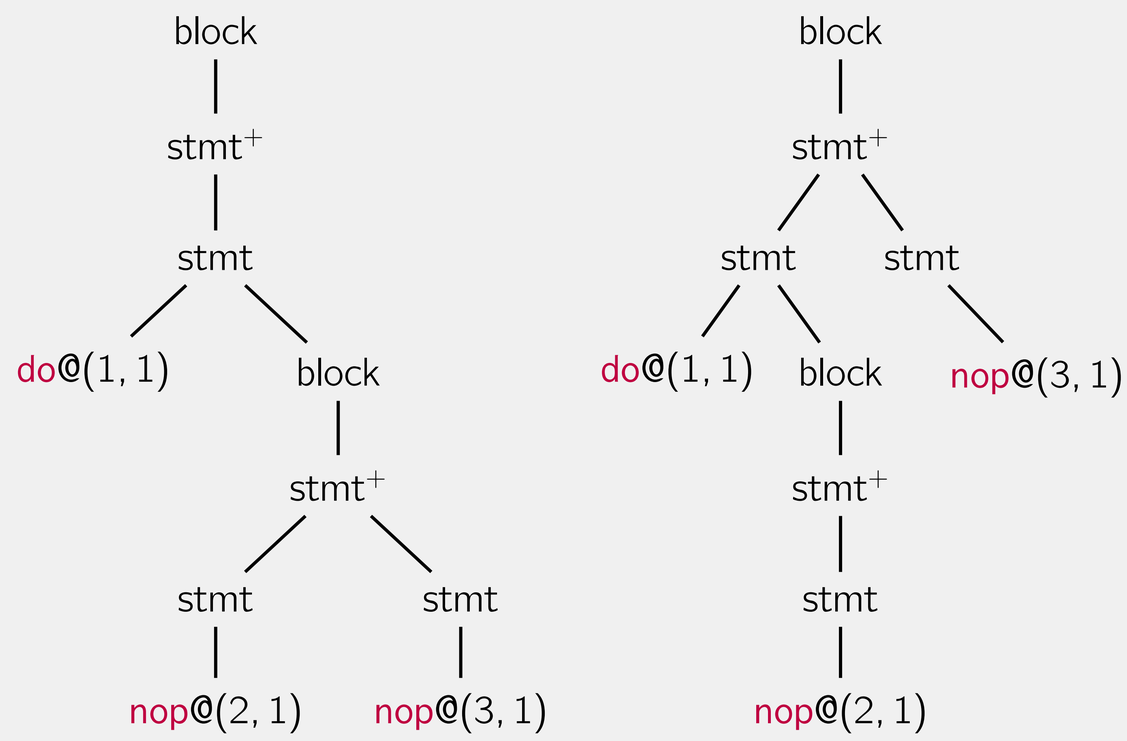
Awesome! But is G_{block} really unambiguous?

Step 2: Run Lamb

It finds a shortest ambiguous sentence (with its parse trees):

```
do
nop
nop
```

(a) ambiguous sentence



(b) parse tree t_1 (c) parse tree t_2

Users can fix the ambiguity issue manually with the aid of the produced parse trees.

Step 3: Understand the cause of ambiguity

It is insufficient to tell whether the second nop statement belongs to the do-block or the top-level block, even with the presence of the alignment constraint.

Step 4: Resolve the ambiguity

A possible solution is to reject this ambiguous sentence via an offside constraint $\cdot^{\triangleright}$ over the do-block:

$$\text{block} \to \|\text{stmt}\|^+$$
$$\text{stmt} \to \text{nop} \mid (\text{do block})^{\triangleright}$$

Step 5: Check the refined grammar again

Lamb no longer finds any ambiguous sentence within a length of 20: that is, bounded unambiguous!

More...

Check our website

Read our paper

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62072267 and Grant No. 62021002).