COMS W4901

Adding New Indentation Rules to SSLANG

Anjali Smith
Avighna Suresh

5/14/23

Abstract

SSLANG (Sparse Synchronous Language) is a functional programming language built on top of the Sparse Synchronous Model, a runtime system proposed by Professor Stephen Edwards and John Hui that allows precise timing control. This semester, we worked with the language design team, led by John Hui, to implement indentation rules in SSLANG. We started this project with the goal of fixing a bug in the lexer, but shifted our focus to implementing new indentation rules that were collectively agreed on by the SSLANG research team. This project required an understanding of Alex, SSLANG syntax, and the lexing stage.

1 Motivating Example:

The following SSLANG program is taken from an issue that Emily Sillars opened up on Github. The program should print out the character 'A', however, it throws an indentation error instead.

```
type Tree
  TwoChildren Tree Tree
  Leaf
  OneChild Int Tree

main ( cout : & Int ) -> () =
  let d = Leaf
  let zzz =
    match d
          Leaf = 65
          _ = 66

  after 10 , (cout : & Int) <- zzz
  wait (cout : & Int)
```

The reason the above program failed to compile with the previous lexer, is because the margin of the line with the match statement, which is the start of a new block, is at a lower indentation than the margin of the previous context, which is the start of the "zzz" in the line above the match statement. To better explain the work that we did this semester, we will begin by explaining the indentation rules of the old scanner, the bug that caused the above program to fail, our first attempted solution, and our finalized indentation rules.

2 Old Indentation Rules

The scanning stage of the lexer is implemented in the Scanner.x file. In this file, the lineFirstToken function is called when the scanner encounters the first token of a new line and the blockFirstToken function is called when the scanner encounters the first token of a new block.

2.1 lineFirstToken Function

Every token in a SSLANG program has an associated context, and the scanner maintains a stack of contexts which gets updated as the program is scanned. The function lineFirstToken looks at three important variables to determine the scanner's next course of action: the current context which is referred to as "ctx", the current margin which is referred to as "tCol", and the margin of the current context which is referred to as "margin". When the first token of a new line is encountered, there are three possible contexts that are important to the scanner: ExplicitBlock, ImplicitBlock, and PendingBlockNL.

If the token is in the ExplicitBlock context, meaning the token is within a block enclosed by braces, the scanner simply resumes scanning.

If the token is in an ImplicitBlock context, meaning the token is within a block that is not enclosed by braces, the scanner can take three possible courses of action depending on the line margin of the current token. If the line margin of the current token is greater than the line margin of the start of the context, continue scanning. If the line margin of the current token is equal to the line margin of the current context, insert a separator if needed, and continue scanning. Otherwise, the scanner believes that the block has ended so it emits a closing brace, pops a context, and recursively restarts the lineFirstToken function to continue to check if other ending blocks need to be closed.

If the token is in a PendingBlockNL context, meaning the current token is starting a new block on a new line, the scanner can take two possible courses of action depending on the current token's line margin. If the current token's line margin is less than or equal to the context's margin, then the scanner throws an indentation error. Otherwise, the scanner pops the PendingBlockNL context, pushes the ImplicitBlock context, and then emits the opening brace for the new block right after the current token to start a new implicit block.

2.2 blockFirstToken Function

The blockFirstToken function carries out the necessary checks and actions for the first token in a block. It first ensures that the current context is PendingBlock, and throws an internal error on any other context type. It then checks to ensure that the margin of the current token is greater than the block margin. If it is not, the function throws an indentation error. It wraps up by pushing the ImplicitBlock context to the stack and emitting an opening brace to start a new block at the current token.

2.3 Scanner Bug Explained

The bug that caused Emily's issue was present in the blockFirstToken function, at the comparison of the current token's margin with the context's margin in the pendingBlock context.

```
let {zzz =
  {match d
```

The above code snippet has been edited to include opening braces that indicate the context's margin, which is the start of the "zzz", and the current token's margin, which is the start of the match keyword. Since the current token's margin is at a lower indentation level than the context's margin, the scanner threw an error. However, we wanted this code to compile. Our first attempt to solving this bug was to add a new variable to the scanner: a previous line margin.

3 Attempted Solution: lastLineMargin

We believed that by adding a previous line's margin variable, we could compare the current token's margin with the previous line's margin rather than the context's margin. In order to add a variable for the previous line's margin, we added a lastLineMargin data constructor to the AlexUserState datatype:

```
-- | The state attached the 'Alex' monad; scanner maintains a stack of contexts.
data AlexUserState = AlexUserState
  { usContext :: [ScannerContext] -- ^ stack of contexts
  , commentLevel :: Word          -- ^ 0 means no block comment
  , lastCtxCode :: Int            -- ^ last seen scanning code before block
  , lastLineMargin :: Int         -- ^ previous line's margin
  }
```

We then wrote the two following helper functions:

```
-- | Extracts the margin of the previous line from the AlexUserState
extractMargin :: AlexUserState -> Int
extractMargin (AlexUserState _ _ _ m) = m

-- | Update the previous line margin
updateMargin :: AlexInput -> Alex ()
updateMargin i = do
  let c = alexColumn $ alexPosition i
  st <- alexGetUserState
  alexSetUserState $ st { lastLineMargin = c }
```

Using these functions, we updated the LineStart and the BlockLineStart Alex rules to update the lastLineMargin variable whenever a new line is encountered.

After adding support for the lastLineMargin variable, we then updated the blockFirstToken function to use the lastLineMargin variable to check whether or not the current token's margin is less than the previous line's margin in the PendingBlock context. If this check evaluates to true, the scanner throws an indentation error. Even though this change fixed the specific indentation bug that Emily pointed out, Professor Edwards and John Hui decided that they wanted to bring in the opinions of the greater group to collectively agree on a new set of indentation rules. More specifically, they wanted to clear any ambiguity about how "outdenting", or starting a new line of SSLANG code at a lower indentation than the previous line, should be handled by the lexer.

4 New Indentation Rules

4.1 Overview

This section describes the general indentation rules that SSLANG programmers must follow when writing code. When writing a new line of code, SSLANG programmers should be aware of whether they are starting a new block, continuing code within an explicit block, or continuing code within an implicit block.

If their new line of code is within an explicit block, programmers are free to indent their program in any way they like.

If their new line of code is starting a new block, they must ensure that the start of the line is at the same or greater indentation than the previous line.

If their new line of code is within an implicit block, the margin of this line must not be both less than the previous line's margin and greater than the context's margin.

The following section will explain these rules in detail by going over their implementation, as well as different examples of acceptable indentation in SSLANG programs.

4.2 Implementation

The new indentation rules were implemented by updating the lineFirstToken function, which is called when the first token of a new line is scanned. The rules rely on three margin-related variables: the current token's margin, the current context's margin, the previous line's margin. Depending on the token's current context and margin-related variables, different courses of actions must be taken by the scanner to scan the SSLANG program.

4.2.1 ExplicitBlock Context

As before, if the current token is in an explicit block, continue scanning.

4.2.2 ImplicitBlock Context

If the current token is in an implicit block, there are three possible cases: the current margin is equal to the context's margin, the current margin is greater than the context's margin, and the current margin is less than the context's margin.

If the current token's margin is equal to the context's margin, check if a separator must be inserted into the code and continue scanning. The following code snippets are examples of when the current token is in an implicit block and the current token's margin is equal to the context's margin.

Example 1:

```
loop f
    g // current token margin > last line margin && = context margin
    h // current token margin = last line margin && = context margin
      x
    f // current token margin < last line margin && = context margin
```

Example 2:

```
loop f
g // current token margin = last line margin && = context margin
```

Example 3:

```
loop loop f
     g // current token margin > last line margin && = context margin
```

Example 4:

```
f
    x
g // current token margin < last line margin && = context margin
```

If the current token's margin is less than the context's margin, insert a closing brace in the code and pop a context. Then, recursively restart the lineFirstToken function on the current token with the previous context's margin to check if any other blocks need to be closed or if an indentation error exists. The following code snippet is an example of a SSLANG program where the current token's margin is less than the context's margin.

```
foo
    bar
baz // current token's margin < context's margin
```

If the current token's margin is greater than the context's margin, the scanner must compare the current token's margin to the margin of the previous line. If the current token's margin is less than the margin of the previous line, the scanner throws an indentation error. Otherwise, the current token is a line continuation, and the scanner will continue to scan. The following code snippets are examples of when the current token's margin is greater than the context's margin and greater than or equal to the previous line's margin.

```
f
    x // current token's margin > previous line's margin
    y // current token's margin = previous line's margin
```

4.2.3 PendingBlockNL Context

If the current token is in the PendingBlockNL context, meaning the current token is starting a new block on a new line, the scanner compares the current token's margin with the previous line's margin. If the current token's margin is less than the previous line's margin, throw an indentation error. Otherwise, we have just encountered the first token of the new block so we must transition from the PendingBlockNL context to the ImplicitBlock context. We implemented this by popping the context off of the context stack, inserting an opening brace, and pushing the ImplicitBlock context on the context stack.

5 Testing

We added Emily's test case to the regression tests, named indentation-test.ssl. We also updated the tests in the ParseBlockSpec.hs to reflect the updated indentation rules. The following code snippet shows the updated test:

```
it "allows starting block to be at same indentation" $ do
-- This is bad style but should still parse
shouldParse [here|
  main (clk : &Int) =
  loop
    wait clk
|]
shouldParse [here|
  main (clk : &Int) =
    loop
    wait clk
|]

it "parses singleton let-blocks without needing extra indentation" $ do
shouldParse [here|
  main (clk : &Int) =
    let x =
      1
    x
|]
```

6 New Issue: Memory Leak on Wrong Main Function Signature

While working on this project, we found that when SSLANG programmers implement a function with the wrong main function signature, their program compiles and leaks memory rather than throwing a communicative compilation error. For example, the following program:

```
main ( cout : & Int ) -> () =
    after 10 , (cout : & Int) <- 65
    wait (cout : & Int)
```

produces the following error in the runtests.log file:

```
###### Testing memory-leak-main-func
stack exec sslc -- tests/memory-leak-main-func.ssl > out/memory-leak-main-func.c
sizeof(struct ssm_mm) = 4
page size 4096
pages allocated 2
objects allocated 4
objects freed 3
live objects 1
4 pools
pool   0: pages   0  block-size    16  free-blocks     0
pool   1: pages   1  block-size    64  free-blocks    63
pool   2: pages   1  block-size   256  free-blocks    16
pool   3: pages   0  block-size  1024  free-blocks     0

FAILED: 1 live objects leaked at the end
###### FAILED
```

While this bug has not been fixed, it has been noted under issue 141 on Github.

7 Final Outcome

We merged our code into main which included our changes to the Scanner.x file, the new regression test, and the updated ParseBlockSpec.hs file.

8 Next Steps

Now that the research group has agreed on a working set of indentation rules, it would be useful to create an indentation guide for SSLANG programmers. We should collectively decide which details should be included in the guide when listing out each rule. For example, we may want the guide to share which keywords start a new block on the same line or a new line, but we may want to hide more specific details of the lexer implementation to keep the guide simple.

References

1. Bhattacharya, Jyotirmoy. Alex and Happy. Leanpub, 2014. leanpub.com, https://leanpub.next/alexandhappy.
2. Stephen A. Edwards and John Hui. The Sparse Synchronous Model. In Forum on Specification and Design Languages (FDL), Kiel, Germany, September 2020.