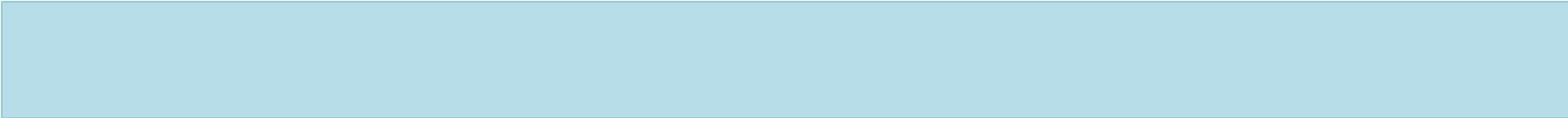


Lecture 14

CS 131: COMPILERS

Announcements

- Midterm: November 19th
- HW4: OAT v. 1.0
 - Parsing & basic code generation
 - Release soon



Debugging parser conflicts.
Disambiguating grammars.

MENHIR IN PRACTICE

Practical Issues

- Dealing with source file location information
 - In the lexer and parser
 - In the abstract syntax
 - See `range.ml`, `ast.ml`
- Lexing comments / strings

Menhir output

- You can get verbose ocaml yacc debugging information by doing:
 - `menhir --explain ...`
 - or, if using dune, adding this stanza:
(menhir
 (modules parser)
 (flags --explain --dump))
- The result is a `<basename>.conflicts` file that contains a description of the error
 - The parser items of each state use the `'` just as described above
- The flag `--dump` generates a full description of the automaton
- Example: see `start-parser.mly`

Precedence and Associativity Declarations

- Parser generators, like menhir often support precedence and associativity declarations.
 - Hints to the parser about how to resolve conflicts.
 - See: `good-parser.mly`
- Pros:
 - Avoids having to manually resolve those ambiguities by manually introducing extra nonterminals (as seen in `parser.mly`)
 - Easier to maintain the grammar
- Cons:
 - Can't as easily re-use the same terminal (if associativity differs)
 - Introduces another level of debugging
- Limits:
 - Not always easy to disambiguate the grammar based on just precedence and associativity.

Example Ambiguity in Real Languages

- Consider this grammar:

$S \mapsto \text{if } (E) S$

$S \mapsto \text{if } (E) S \text{ else } S$

$S \mapsto X = E$

$E \mapsto \dots$

- Is this grammar OK?

- Consider how to parse:

$\text{if } (E_1) \text{ if } (E_2) S_1 \text{ else } S_2$

- This is known as the “dangling else” problem.
- What should the “right” answer be?
- How do we change the grammar?

How to Disambiguate if-then-else

- Want to rule out:

if (E₁) if (E₂) S₁ else S₂ }

- Observation: An un-matched 'if' should not appear as the 'then' clause of a containing 'if'.

S \mapsto M | U // M = "matched", U = "unmatched"
U \mapsto if (E) S // Unmatched 'if'
U \mapsto if (E) M else U // Nested if is matched
M \mapsto if (E) M else M // Matched 'if'
M \mapsto X = E // Other statements

- See: `else-resolved-parser.mly`

Alternative: Use { }

- Ambiguity arises because the 'then' branch is not well bracketed:

`if (E1) { if (E2) { S1 } } else S2 // unambiguous`

`if (E1) { if (E2) { S1 } else S2 } // unambiguous`

- So: could just require brackets
 - But requiring them for the else clause too leads to ugly code for chained if-statements:

```
if (c1) {  
    ...  
} else {  
    if (c2) {  
  
    } else {  
        if (c3) {  
  
        } else {  
  
        }  
    }  
}
```

So, compromise? Allow unbracketed else block only if the body is 'if':

```
if (c1) {  
  
} else if (c2) {  
  
} else if (c3) {  
  
} else {  
  
}
```

Benefits:

- Less ambiguous
- Easy to parse
- Enforces good style



See HW4

OAT V. 1



Untyped lambda calculus

Substitution

Evaluation

FIRST-CLASS FUNCTIONS

“Functional” languages

- Oat (like C) has only top-level functions
- Languages like OCaml, Haskell, Scheme, Python, C#, Java, Swift
 - Functions can be passed as arguments (e.g., to map or fold)
 - Functions can be returned as values (e.g., from compose)
 - Functions *nest*: inner function can refer to variables bound in the outer function

```
let add = fun x -> fun y -> x + y
let inc = add 1
let dec = add -1

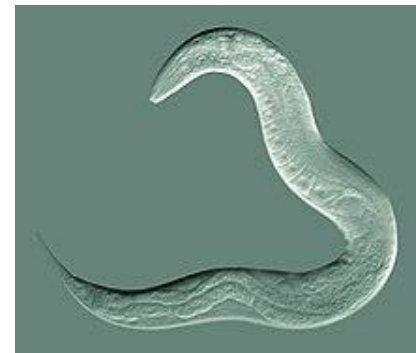
let compose = fun f -> fun g -> fun x -> f (g x)
let id = compose inc dec
```

- How do we implement such functions?
 - in an interpreter? in a compiled language?

(Untyped) Lambda Calculus

- The lambda calculus is a *minimal* programming language
 - OCaml: (fun x -> e)
 - lambda-calculus notation: $\lambda x. e$
- It has variables, functions, and function application.
 - That's it!
 - It's Turing Complete(!!)
 - It's the foundation for a *lot* of research in programming languages.
 - Basis for “functional” languages like Scheme, ML, Haskell, etc.

Lambda calculus is the c. elegans of programming languages. Its minimal (but not too minimal!) form lets us deeply characterize its properties.



c. elegans – with 6 chromosomes, fully sequence DNA, 302 neurons, and extremely well-studied life cycle is a "model organism" used in biology.