CS3342: Lecture Notes

Lecture notes will be added here as they become available. There may be changes to existing slides, so it is best to download them before each class.
Here is a weekly plan of what is being taught:

week			TOPIC		
			1h	2h	3h
1	11-Jan	12-Jan	Outline	Intro	Intro
2	18-Jan	19-Jan	Syntax - Reg.Ex.	Syntax - CFG	Syntax - FA
3	25-Jan	26-Jan	Scanning	Parsing - intro	LL Parsing
4	01-Feb	02-Feb	First, Follow	LL(1)	LR Parsing
5	08-Feb	09-Feb	LL(1) vs SLR(1) vs unambig	Semantics	Semantics
6	15-Feb	16-Feb	Names	Names	Flow
	22-Feb	23-Feb	Reading week		
7	01-Mar	02-Mar	Midterm		Types
8	08-Mar	09-Mar	Types	Types	OO - Dynamic Method Binding
9	15-Mar	16-Mar	λ-calculus	λ-calculus	λ-calculus - modelling
10	22-Mar	23-Mar	Functional Programming	Scheme	Scheme
11	29-Mar	30-Mar	Predicate Calculus	Predicate Calculus	Logic Programming
12	05-Apr	06-Apr	Prolog	Prolog Control Algorithm	Prolog Trees

The corresponding readings from the textbook are indicated after each slide set. **The material taught in class is required for exams.** The readings from the textbook are used as support.

- Introduction
  - Readings: Chapter 1
- Syntax: Scanning, LL parsing, LR parsing
  - Topics: Regular expressions, Context-free grammars, Derivations, Parse trees, Ambiguity, Lexical analysis (scanning), DFA, Top-down (LL) parsing (recursive descent, table-driven), First, Follow, Predict sets, LL(1)-grammars, Bottom-up (LR) parsing (table-driven), SLR(1)-grammar, Characteristic finite state machine
  - Readings: Chapter 2
- Semantic Analysis
  - Topics: Attribute grammar, Parse tree annotation, Synthesized attributes, Inherited attributes, S-attributed grammar, L-attributed grammar
  - Readings: Chapter 4 (4.1-4)
- Names, Scopes, and Bindings
  - Topics: Storage allocation (static, stack, heap), Garbage collection, Referencing environment, Scope (static, dynamic), Binding (shallow, deep), First-class functions, Lambda expressions
  - Readings: Chapter 3 (without 3.3.4-5, 3.5, 3.6.3, 3.7, 3.8)
- Control Flow
  - Topics: Infix, prefix, postfix expressions, Precedence, Associativity, Side effects, Value model, Reference model, Short-circuit evaluation, Iterators, Recursion vs iteration, Tail recursion, Lazy evaluation
  - Readings: Chapter 6 (without 6.1.2 (after references and values), 6.1.3-4, 6.2-3, 6.4.2, 6.5.1-2, 6.5.4-5, 6.7) and Sections 9.3.1-2
- Types
  - Topics: Type systems, type checking, polymorphism, arrays, pointers, lists, garbage collection
  - Readings: Chapters 7, 8 (7.2.1-3, 7.3.1, 8.2, 8.4-6)
- Object-Oriented Programming
  - Topics: classes, encapsulation, inheritance, constructors/destructors, virtual methods, dynamic method binding
  - Readings: Chapter 10 (without 10.2.3-5, 10.4.4, 10.5-7)

- Lambda Calculus
    - Topics: λ-calculus, λ-expressions, syntactic rules, free and bound variables, substitution, computing with λ-terms, cal-by-value and call-by-name reductions, modelling integers and booleans
    - Readings: Section 11.7 (Lambda-calculus)
    - Lambda reduction examples
- Functional Programming
    - Topics: Scheme
    - Readings: Chapter 11 (without 11.4, 11.5.2)
    - DFA simulation example
- Predicate Calculus
    - Topics: Predicate calculus, clausal form, Horn clauses, resolution
    - Readings: Section 12.3 (Predicate calculus)
- Logic Programming
    - Topics: Prolog
    - Readings: Chapter 12
    - Prolog tree examples

- Sample exams
    - Midterm Exam Sample (Solution)
    - Final Exam Sample (Solution)

- More practice problems
    - A1 (A1_sol)
    - A2 (A2_sol)
    - A3 (A3_sol)
    - A4 (A4_sol)

- Lambda Calculus
  - Topics: λ-calculus, λ-expressions, syntactic rules, free and bound variables, substitution, computing with λ-terms, cal-by-value and call-by-name reductions, modelling integers and booleans
  - Readings: Section 11.7 (Lambda-calculus)
  - Lambda reduction examples

rule:

1. left-associative

   $x\ y\ z \Rightarrow (x y) z$

2. application has higher precedence

   $\lambda x. A B \Rightarrow \lambda x. (A B)$ * NOT $(\lambda x. A) B$.

3. consecutive abstraction:

   $\lambda x_1 x_2 \cdots x_n . e \Rightarrow \lambda x_1 (\lambda x_2 (\cdots (\lambda x_n . e))))$

$\lambda ab . abc\ (de)$

$\Rightarrow$ a, b are bound variables

$\Rightarrow$ d, e are free variables

call-by-name: leftmost outermost

call-by-value: leftmost innermost

* sometimes call-by-value would be trapped in inf loop

modeling:

$T \equiv \lambda x \lambda y . x$

$F \equiv \lambda x \lambda y . y$

integers:

$0 \equiv \lambda f . \lambda c . c$

$1 \equiv \lambda f . \lambda c . (f c)$

$2 \equiv \lambda f . \lambda c . (f (f c))$

$3 \equiv \lambda f . \lambda c (f (f (f c)))$

$\vdots$

$NOT \equiv \lambda x ((x F) T)$

$AND \equiv \lambda x \lambda y . ((x y) F)$

$OR \equiv \lambda x \lambda y . ((x T) y)$

- Functional Programming
  - Topics: Scheme
  - Readings: Chapter 11 (without 11.4, 11.5.2)
  - DFA simulation example

(define a 2)

(define (multiply x y) (* x y))

(lambda (x y) (* x y)) <= anonymous function

car: give first element (head)

cdr: give rest of the element (tail)

cons: add element to a list

list: create a list (list 'a 'b 'c) => (a b c)

let: binding. e.g. (let ((a 2) (p +) (b 4)))

P A B => 8

- Predicate Calculus
    - Topics: Predicate calculus, clausal form, Horn clauses, resolution
    - Readings: Section 12.3 (Predicate calculus)

normal form:

1. remove ->, <->
2. move negations inward using De Morgan's law
3. using Skolemization, pick a random variable to eliminate "∃"
4. pull universial quantifiers to the front
5. drop those universial quantifiers at the front
6. convert to conjunctions of disjunctions.

   i.e. $(A \vee \neg B) \wedge (C \vee D \vee E) \wedge (\quad) \cdots\cdots$

horn clause:

$(L_1 \wedge L_2 \wedge L_3 \cdots \wedge L_n) \rightarrow H$

$L_1 \cdots L_n$ could be either positive or negative, but H must be positive

one specific case:

$\neg Q_1 \vee \neg Q_2 \vee \neg Q_3 \cdots \vee \neg Q_n \vee P \equiv \neg(Q_1 \vee Q_2 \cdots \vee Q_n) \vee P$

$\equiv (Q_1 \vee Q_2 \cdots \vee Q_n) \rightarrow P$

Resolution example

```
student(X) :- resident(X).
student(X) :- takes(X, Y), class(Y).
resident(john).
takes(mark, 3342).
class(3342).

?- student(john).
true
```

- Resolution (add negation of query):

(¬resident(X) ∨ student(X)) ∧
(¬takes(Y, Z) ∨ ¬class(Z) ∨ student(Y)) ∧
resident(john) ∧
takes(mark, 3342) ∧
class(3342) ∧
¬student(john)

same line: ∨

diff line: ∧

query: ¬

* keep variables free!

- Logic Programming
    - Topics: Prolog
    - Readings: Chapter 12
    - Prolog tree examples