Carleton University, Computer Science, Winter 2018

Comp 3002 Compiler Construction
Course Outline

**Introduction**:

A quick review of the components of a compiler: scanner, tree building parser, tree walker for code generation. A quick review of the process of designing a language for which a compiler is to be built.

A quick 1 lecture preview where we design a parser transduction grammar and a scanner transduction grammar for a simple expression language, feed it into a table building program to generate the tables, and implement a compiler for it.

**Scanner/Parsers**

A discussion of the details of a table driven scanner/parser and how it works.

**The infrastructure for parsers/scanners**

Detailed discussion of finite state machines and regular expressions and their use in transduction grammars. This includes operations such \*,+,&,|,-, and . (concatenation).

**Converting a transduction grammar with regular expression right parts into a transduction grammar with FSM right parts.**

This is a tree walking process akin to code generation but used instead in the context of grammar conversion.

**Theoretical underpinnings for transduction grammars**

Regular grammars, context free grammars, regular right part grammars, parse trees versus abstract syntax trees, handles, regular right part transductions grammars, LL(k) versus LR(k) grammars.

**Constructing tables for regular right part grammars**

 More than shift-reduce tables; specifically, how to construct readahead FSMs for finding the right end of a handle, readback FSMs for finding the left end of a handle, and semantic action states for tree building along with their conversion into table format for use by scanner/parsers. Overview of the process followed by detailed coverage.

**Symbol tables**

Techniques for dealing with scoping in typical programming languages.

**Code Generation Basics**

 Machine versus virtual machine instruction sets. Code generation basics for virtual machine instruction sets. The distinction between expression contexts which requires a value on the virtual machine stack versus statement contexts which do not with several examples of language constructs that can be in used in both contexts.

**More advanced code generation**

Dealing with if statements, while loops, for loops. Dealing with short circuit boolean operations such as &&, ||, and ! which DO NOT require the use of And, Or, or Not virtual machine instructions.

**Other Topics**

Chain reduction optimizations for parsers. Non-canonical parsers.