COMP26020 Programming Languages and Paradigms (70)

Programming Paradigm defines a fundamental style of programming (how programmer write as well as data and computations). Can be used to classify programming languages. Some paradigms are (much) better suited than others to solve a given kind of software engineering problem.

Imperative Paradigm describes sequence of statements manipulating the program state. Assembly is *unstructured*, while most modern languages are *structured* and have loops, conditionals, and procedures.

Declarative Paradigm asks programmer describes the meaning/result of computations. *Declarative Functional* calls and compose functions to describes the program, have first-class functions, use and pure functions have no side-effects.

First-class Function is a function that can be passed as parameter to other functions, and returned from other functions.

A language must have at least one paradigm, while many languages are multi-paradigm.

Static Initialisation uses braces. Something like `ms m = {4, 2};`

Preprocessor is called in general under the hood by the compiler and works on source files, performing textual transformations and producing source files as output. This is done before the actual compilation phase.

System Call Number is stored in register `%rax`
Parameters to the system call are stored in `%rdi`, `%rsi`, and `%rdx` in this order.
Return Value of a system call is placed in `%rax`.

Double Exclamation Mark Operator (!!) in Haskell is used to access the element at a specified index of a list or a tuple.

Strictness of an Argument in Haskell refers to whether or not the argument is evaluated before it is used by the function. A strict argument is evaluated before it is used, while a lazy argument is not evaluated until it is needed. It can be specified by the programmer using the `!` operator, or can be inferred by compiler if value is being used in calculation.

>> Operator is used to sequence two computations, but would discard the return value.
>>= Operator is used to bind two computations, and would pass the return value as an argument.

Do Statement can connect a lot of computations in lines, but to have a local variable it would be `n<-value`.

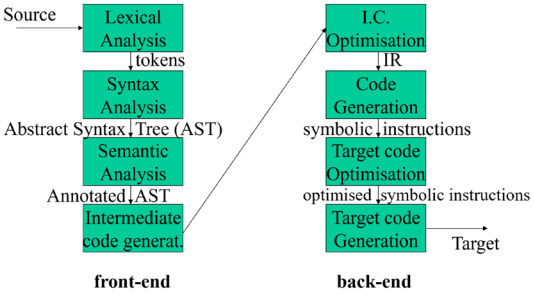
Ambiguity exists if a grammar produces more than one parse tree for some sentence, or has more than one leftmost / rightmost derivation for a single sentential form.

Top-Down Parsing begins with the starting symbol of the grammar (parse tree root), and repeat until the input string has been (fully) matched.

LL(1) Property means parser can look ahead one symbol in the input stream to determine the next production to use. It is a top-down parsing approach.
LR(1) Grammar (i) isolate the handle of each right-sentential form, and (ii) determine the production by which to reduce, by scanning the sentential form from left-to-right, going at most 1 symbol beyond the right-end of the handle. It is a bottom-up parsing approach.

Reduce is the action that would change the format of the rightmost elements to the higher form.
Shift happens when there is no reduce to do, and push the next element to the stack of elements.
ACTION-GOTO Table is a reference tool used by *LR(1)* parsers. It first preforms the ACTIONS based on the next elements, or use GOTO.

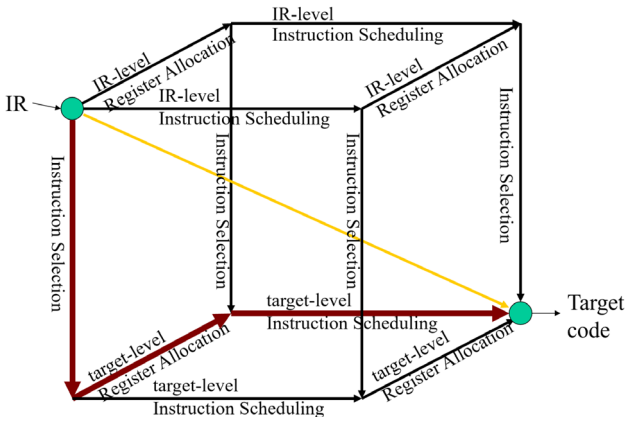
Thomspon's Construction is a method transform regular expressions into equivalent *nondeterministic finite automata*.
Subset Construction is a standard method for converting a *NFA* into a *deterministic finite automaton (DFA)*.
Hopcroft Algorithm is used in compilers to minimize *DFA*s by iteratively merging states in a DFA that are indistinguishable. It merges two states each time.



Common subexpression elimination: An expression, say x+y, is redundant iff along every path from the procedure's entry it has been evaluated and its constituent subexpressions (x, y) have not been redefined.
Copy propagation: After a 'copy' statement, x=y, try to use y as far as possible.
Constant propagation: Replace variables that have constant values with these values.
Constant folding: Deduce that a value is constant, and use the constant instead.
Dead-code elimination: A value is computed but never used; or, there is code in a branch never taken (may result after constant folding).
Reduction in strength: Replace an expensive operation with a cheaper operation
Loop-invariant code-motion: Detect statements inside a loop whose operands are constant or have all their definitions outside the loop - move out of the loop.

Basic Block is a segment of straight-line (i.e., branch-free) code.
Interference exists when two values are simultaneously live when an operation occurs.

Top-down Register Allocation reserve registers for the most frequently used values. All other values are loaded from / stored to memory when needed. Sometimes value may not be active in all segments of code.
Bottom-up register allocation (Best's algorithm) keeps a pool of registers; assign one register when a value is initialised (start of the *live range*); return register to the pool at the end of the live range.
Register Allocation via Graph Colouring construct *live ranges*, build *interference graph*, and try to construct a *k-colouring* of the graph and then map to physical registers.
Instruction Scheduling is completed by building precedence graph, compute priority function for the nodes, and then use list scheduling to construct a schedule by pop out a ready operation and schedule it.



Sethi-Ullman Labelling Scheme assigns labels to the nodes of an AST to determine the minimum number of registers required to evaluate the expression represented by the AST, from leaves to root.
Short-Circuit Evaluation is a technique used by compilers to evaluate Boolean expressions - only evaluates the parts of the expression that are necessary to determine the overall value of the expression.

Concurrency is running parts of the same computation at the same time. It breaks our assumptions about how code works, especially atomicity and ordering.

Deadlock: A cycle of processes where all wait for the next process in the cycle to leave a critical section.
Livelock: A cycle of processes where all try to enter a critical section but they all fail.
Starvation: One or more process cannot progress because they need a resource that they cannot acquire