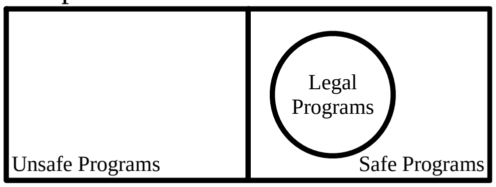
Lecture Notes 8

Type Checking

- Dynamic Checking – Type information is maintained and checked at runtime
- Static Checking – Type information is maintained and checked at translation time
- Type Inference – Types of expressions are determined by types of subexpressions
- Compatibility – Two different types that can be combined
  - Assignment Compatibility – Type correctness of assignment
  - L-value or reference assigned to R-value or dereferenced
- Implicit Types – Types that have the type implied (not explicitly stated)

Strongly Typed

- Strongly-Typed Languages – All (unintentional) data-corrupting errors are caught at earliest point (mostly translation time)
  - Safe vs. Unsafe Programs – Unsafe programs have data-corrupting errors
  - Legal Programs – Subset of safe programs that translator will accept
  - Exmples: Ada



- Weakly Typed Languages – Languages that have “loopholes” in the strongly typed system
  - Examples: C & C++
- Untyped or Dynamically Typed Languages – Type is determined at runtime
  - Examples: Scheme, Smalltalk, Perl, Python

Type Equivalence

- Structural Equivalence – Equivalent if have identical structure, they will have the same set notation sets
- Type Names – Name associated to a constructed type
- Anonymous Types – No name associated to the constructed type
- Name Equivalence – Only equivalent if type names are same
- Aliases – Create equivalent types with different names
- Declaration Equivalence – Equivalent if declaration leads back to same type

Theory and Data Types

- Data Types as Sets
  - Set operations can be applied: union, powerset, etc.
  - Type Constructors – Creating types through set operations
- Cartesian Product
  - $U \times V = \{(u,v) \mid u \in U, v \in V\}$
  - Record or Structure Construction – Example of Cartesian Product

    ```
    struct IntChar{
        int i;
        char c;
    };
    ```

    - `IntChar` is the Cartesian Product `int` × `char`
  - Projections: Component Selector or Structure Member Operator
    - Projection functions p_1: $U \times V \rightarrow U$, p_2: $U \times V \rightarrow V$
    - $p_1((u,v)) = u$, $p_2((u,v)) = v$

      ```
      struct IntChar x;
      x.i;
      ```

    - `x.i` is the project p_1(`x`)
  - Tuples (ML: type IntCharReal = int * char * real;)
  - Class – Cartesian Product with Functions
    - Member Functions or Methods
- Union
  - $U \cup V$
  - union (C/C++)

    ```
    enum Disc {IsInt, IsReal};
    struct IntOrReal{
        enum Disc which;
        union{
            int i;
            double r;
        } val;
    };
    ```

  - Anonymous Union (C++ same as above without val)
  - Variant Record (Ada)

    ```
    type Disc is (IsInt, IsReal);
    type IntOrReal (which: Disc) is
    record
        case which is
            when IsInt => i: integer;
            when IsReal => r: float;
        end case;
    end record;
    ```

- Subset
  - $u \subset U$

    - Subtype (Ada) – Range is specified with lower and upper bounds
    - Inheritance – Operations are inherited from parent set
        - Can’t usually specify which operations
- Map or Partial Map
    - $U \rightarrow V$
    - Array Type or Sequence Type (Ordinal)
        - Index Type U
        - Component Type V
        - Vector or List (Functional Languages)
    - Function Type
        - Function Pointers (C/C++)