1 General

Chomsky Hierarchy The Chomsky Hierarchy describes what is required to read and understand a specific language. Lexer, Parser and Semantic Checker reads the Language while the DEA, Push down Automata and Bounded Turing Machine runs the application.

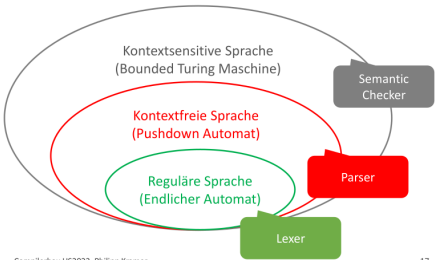

Figure 1: Chomsky Hierarchy

2 Parser

Parser Categories

- first letter
  L read from left to right
  R read from right to left
- second letter
  L Left-most expansion (top-down parser)
  R right-most reduction (bottom-up parser)
- number in parentheses
  – number of symbols look ahead

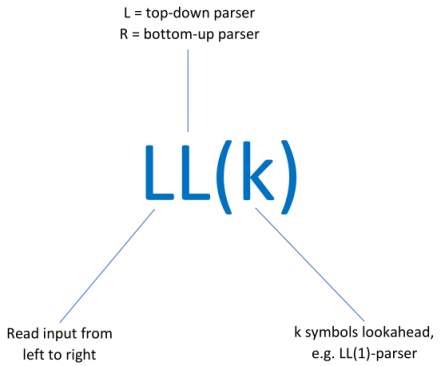

Figure 2: Parser notation

3 Checker

Symbol Table The Symbol Table is a data structure produced by the Semantic Checker. The Symbol table is used to manage all declaration in a program source code.
The symbol table contains a mapping from *Symbol* to the *AST* node (`node = symbolTable.getDeclarationNode(symbol);`)

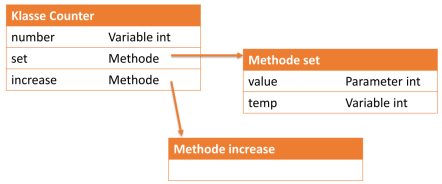

Figure 3: Example Structure for a Symbol Table

4 VM

Ancestor Table The ancestor table is used to check if a class inherits from another in constant time. In Figure 4 you see an example how such an ancestor table is constructed. The last element in the table for each class is the reference to itself. However, this only works for **Single-Inheritance**.
In languages like Java you don't have to allocate memory for the `object` class, because every class inherits from it (in the ancestor table this would be -1).

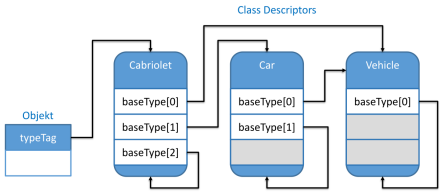

Figure 4: Ancestor Table Example

Type Cast / Checking You have a target type (you want to cast into this type) and a source type (you want to cast from this type).

1. Get the ancestor level from the target type (for example 0). Call the level `x`
2. Check in the ancestor table of the source type if the ancestor at level `x` is the same as the target class

```
private boolean typeTest(
    Pointer instance,
    ClassDescriptor targetType)
    {
     ClassDescriptor sourceType
         = heap.getDescriptor(
        instance);
     if (sourceType ==
         targetType) {
         return true;
     }
     var level = targetClass.
         getAncestorLevel();

     if (sourceClass.
         getAncestorTable().
         length > level) {
          return sourceClass.
              getAncestorTable()[
              level] ==
              targetClass;
     } else {
          return false;
     }
}
```

Virtual Table Each type has a Virtual Method Table (vtable). The vtable describes which virtual method should be called when dynamic dispatch is performed. In a VM the vtable often points not directly to the function but to the method descriptor.
However, this only works with **Single-Inheritance**.

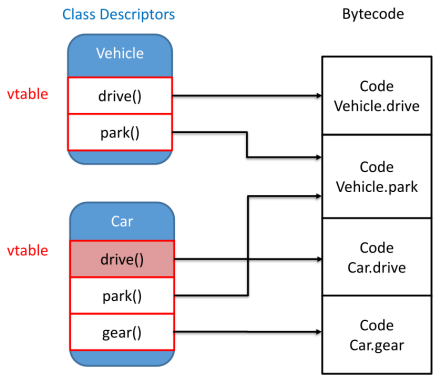

Figure 5: VTable example

5 GC

Generational GC A Generational GC is an incremental GC because it often does not clean up the whole heap but in generations. For example, you have 3 generations as describe in **??**

If you want to clean up *Gn* you must extend your root set with all references from the latter generations which are pointing into *Gn*. If you want to clean up *Gn* you also have to clean up all previous generations.

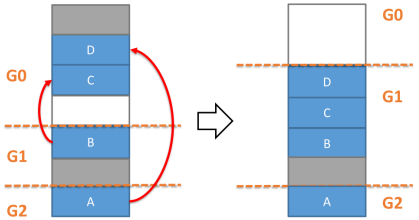

Figure 6: GC of G0

Resurrection A finalizer can revive objects (no garbage anymore). This can happen not only for the current object but also for other objects.

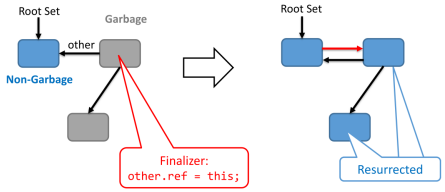

Figure 7: Resurrection Example