DynLang Projekt Dokumentation

Christoph Urlacher

2. September 2021

1 Sytaktischer Zucker

Es waren keine Anpassungen des Interpreters notwendig, nur eine Erweiterung des Lexings/Parsings.

1.1 Lexing

Es wurden neue Token definiert für

- Arithmetische Operatoren (+, -, *, /, %, ...)
- Arithmetische Operatoren mit Assignment (+=, -=, ...)
- Vergleiche (<, <=, ==, ...)
- Logische Operatoren (&&, ||, !)

1.2 Parsing

Arithmetische, Vergleichs- und Logische Operatoren werden im Parser zu Methodenaufrufen der Builtin-Methoden übersetzt. Das Programm ist lauffähig, falls die Traits der entsprechenden Typen diese Methoden Implementieren.

Für Inplace-Operationen wird zusätzlich zum Builtin-Aufruf ein Assignment-Knoten im AST erzeugt.

Klammern von Ausdrücken ist ebenfalls möglich, dafür wird beim Parsing der geklammerte Ausdruck eine Ebene tiefer in den AST eingefügt.

Die Grammatik ist mehrdeutig, Präzedenz wird durch den Parsergenerator behandelt.

2 Primitive Datentypen

Es wurden die Typen Double, String und Boolean hinzugefügt.

2.1 Lexing

Es wurden neue Token definiert für

- String Literale (“Hallo”)
- Boolean Literale (“true”, “false”)
- Double Literale (1.2, 1., .1, -.1, . . .)

2.2 Parsing

Der AST wurde um Expression-Knoten für Double-, String- und Boolean-Literale erweitert (analog zum IntLiteral-Knoten). In der Grammatik können Double/String/Boolean Nichtterminale bei einer basicexpression auftreten.

2.3 Objectmodel

Die Boolean/String/Double/Int Objekte erben vom neuen PrimitiveObject, welches den zugehörigen Trait, die Parents, die Slots und den Wahrheitsgehalt bestimmt.

Der Objectspace bekommt Methoden zum Erzeugen eines neuen Boolean/String/Double.

Für jeden Datentyp wurde ein neuer Builtin-Trait hinzugefügt mit Methoden zum Vergleich/zur Conversion etc (mit den entprechenden Primitives).

2.4 Compiler/Interpreter

Es wurden neue Bytecodes für Literale hinzugefügt. Diese sind größtenteils identisch zum IntLiteral-Bytecode, nur der StringLiteral- und DoubleLiteral-Bytecode speichert den Inhalt in der Symboltabelle.

Die Bytecodes werden analog zu IntLiteral interpretiert, nur bei StringLiteral und DoubleLiteral wird aus der Symboltabelle geladen.

3 Garbage Collection

Es wurde Mark & Sweep verwendet.

3.1 Implementierung

Im Objectmodel wurde der AbstractObject Klasse ein GC-Flag “mark” gegeben.

Im Objectspace werden alle existierenden Objekte in einer Liste gehalten.

Ablauf:

- Alle Markierungen entfernen

- Markieren (rekursiv ausgehend von der Lobby, diese wird also immer markiert)
- Entfernen (aus der Liste im Objectspace)

3.2 Aufruf aus Simple

Es wurde die “gc” Anweisung hinzugefügt: AST-Knoten (GCStatement) und Bytecode (GC)

Der Interpreter ruft im Objectspace die “gc()” auf und legt “true” auf den Stack, da ein Rückgabewert nötig ist.