Blatt 04 – A4.6 Recherche und Diskussion

Modul: Compilerbau — Gruppe: Adrian Kramkowski, Yousef Al Sahli, Abdelhadi Fares, Abdelraoof Sahli

---

A4.6 – Handgeschriebene Parser & ANTLR (Recherche und unsere Einschätzung)

1) Open-Source-Projekte, die handgeschriebene Parser verwenden

Beispiel 1: Lua

Lua verwendet einen handgeschriebenen Scanner und Parser. Früher wurde ein yacc-basierter Parser verwendet, später hat sich das Projekt jedoch bewusst für einen manuell implementierten Parser entschieden.

Genannte Gründe:
- höhere Effizienz und kleinere Implementierung
- bessere Portabilität
- präzisere Fehlermeldungen
- vollständige Kontrolle über das Parsing

Beispiel 2: C#-Compiler (Roslyn)

Der Microsoft C#-Compiler Roslyn nutzt ebenfalls einen handgeschriebenen recursive-descent-Parser. Besonders in IDE-Kontexten (Visual Studio) sind die darauf abgestimmten Anforderungen wichtig.

Genannte Gründe:
- Inkrementelles Re-Parsing (nur geänderte Bereiche werden neu geparst)
- Wiederverwendung bestehender AST-Knoten (spart Zeit & Speicher)
- extrem feine Kontrolle über Fehlermeldungen
- bessere Wartbarkeit in einem langfristigen Projekt

Beispiel 3: Go-Compiler

Auch der Go-Compiler setzt auf handgeschriebene Parser und Lexer.

Genannte Gründe:

- sehr hohe Performance
- direkte Kontrolle über Grammatik & Sprachevolution
- enge Verzahnung mit den restlichen Compiler-Komponenten

Beispiel 4: Ruff (Python-Linter)

Der Python-Linter „Ruff“ ist 2024 von einem generierten Parser auf einen handgeschriebenen Parser umgestiegen.

Genannte Gründe:
- deutlich höhere Geschwindigkeit (über 2x schneller)
- stabilere und gezieltere Fehlererholung
- effizientere Verarbeitung großer Codebasen

Zusammenfassung der Praxisbeispiele

Die Projekte zeigen klar, warum handgeschriebene Parser oft gewählt werden:

- Performancevorteile
- bessere und präzisere Fehlermeldungen
- flexible Fehlerkorrektur / Error-Recovery
- volle Kontrolle über Grammatik und Parsing-Verhalten
- langfristig oft wartbarer als generierte Parser

2) Einschätzung unserer Gruppe: Was spricht für und gegen ANTLR?

Unsere Sicht auf ANTLR – Vorteile
---
1. Schnellere Entwicklung
   - Durch die .g4-Grammatik kann ein funktionierender Parser sehr schnell erzeugt werden.
   - Gerade bei vielen Regeln ist das im Vergleich zum handgeschriebenen Parser eine starke Zeitersparnis.

2. Grammatik ist klar getrennt vom Code
   - Änderungen an der Sprache werden einfach in der Grammatik vorgenommen.
   - Dadurch ist die Wartung insgesamt übersichtlicher.

3. Gute Tool-Unterstützung

- ANTLR bietet Debugger, Visualisierungen und Syntaxhervorhebung.
- Das hilft besonders beim Lernen oder bei Prototypen.

4. Korrektheitsprüfungen
- ANTLR erkennt viele klassische Grammatikfehler automatisch.
- Dadurch spart man sich viel Fehlersuche im Parser selbst.

Unsere Sicht auf ANTLR – Nachteile
---
1. Fehlermeldungen nicht so kontrolliert
- Auch wenn man Error-Listener anpassen kann, ist die Flexibilität geringer als bei einem komplett selbst geschriebenen Parser.

2. Generierter Code ist schwer lesbar
- Der Java-Code, den ANTLR erzeugt, ist komplex und oft unübersichtlich.
- Debugging ist dadurch deutlich anspruchsvoller.

3. Abhängigkeit vom Tool
- Man hängt von den Updates und Entscheidungen des ANTLR-Frameworks ab.
- Für sehr langfristige Projekte kann das ein Risiko darstellen.

4. Grammatik muss an LL(*) angepasst werden
- Manche „natürlichen“ Regeln der Sprache müssen umgeschrieben werden, damit ANTLR sie akzeptiert und effizient verarbeitet.

Unsere Gruppeneinschätzung – direkter Vergleich

Wir finden:

✓ Ein handgeschriebener recursive-descent-Parser ist sinnvoll, wenn
- maximale Kontrolle über Fehlermeldungen benötigt wird
- die Sprache klein oder sehr speziell ist
- Performance besonders wichtig ist
- man Parsing-Strategien exakt anpassen möchte

✓ ANTLR ist besser geeignet, wenn
- die Grammatik groß oder komplex ist
- man schnell Ergebnisse benötigt
- Standard-Fehlerbehandlung ausreicht
- die Sprache häufig geändert wird

Gemeinsames Fazit der Gruppe

Für unser Projekt halten wir einen handgeschriebenen Parser für besonders lehrreich, weil wir damit den Ablauf des LL-Parsing genau verstehen und kontrollieren können.

Gleichzeitig sehen wir ANTLR als sehr nützliches Werkzeug, wenn es später darum geht, größere oder komplexere Sprachen zu implementieren, bei denen Entwicklungsgeschwindigkeit im Vordergrund steht.