Wintersemester 2023/24

Parser Project

Timo Plieth **Jakob Ziechmann**

1 Aufgabe

Bei diesem Parser Projekt handelt es sich um eine freiwillige Übung aus der Universität. Der Parser soll logische Formeln Interpretieren können, welche die Operationen $\wedge$(und), $\vee$(oder), $\neg$(nicht), $\Rightarrow$ (Implikation rechts), $\Leftarrow$(Implikation links) und $\Leftrightarrow$(Äquivalenz) beinhalten können. Dazu soll der Parser nachdem er einen Ausdruck gelesen hat ihn mit dem Quine–McCluskey Algorithmus vereinfachen.

2 Syntax

Die Syntax der logischen Ausdrücke soll den Regeln, welche in der boolschen Algebra definiert sind, folgen mit ein par kleineren Änderungen, damit gegebene Ausdrücke leichter per Tastatur umzusetzen sind. So werden die Operatoren $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftarrow$ und $\Leftrightarrow$ durch folgende Zeichen dargestellt: *, +, !, =>, <=, <=>

2.1 Grammatik

Die erste Aufgabe ist an dieser Stelle eine Grammatik zu definieren, auf welcher die nachfolgenden Systeme und Entscheidungen basieren werden.

```
Ausdruck        -> '(' Ausdruck ')' | Ausdruck InfixOperator Ausdruck |
                    PrefixOperator Ausdruck | Literal | Identifier
InfixOperator   -> '*' | '+' | '=>' | '<=' | '<=>'
PrefixAusdruck  -> '!'
Literal         -> '1' | '0'
Identifier      -> ['a'|...|'z']
```

3 Implementierung

3.1 Lexer

Die erste ebene eines jeden Parsers ist der Lexer oder auch Tokenizer genann. Er hat die Aufgabe einen gegebenen String in seine Einzelteile zu zerlegen. Um das in einem Beispiel zu verdeutlichen betrachtn wir den String $(p * q) \Rightarrow a$, welcher von dem Tokenizer in '(', 'p', '*', 'q', ')', '=>' 'a' zerlegt wird. Wichtig dabei ist, dass der Tokenizer immer das längste valide Token bilden möchte, sonst wird nämlich aus abc 'a', 'b', 'c' und nicht 'abc'.

Wir lösen das in unserm Projekt mit einem Deterministischen Automat, welcher die Sprache aller gültigen Tokens akzeptiert. Die Idee ist and dieser Stelle, dass wir den Inputstring Zeichen für Zeichen lesen, bis wir in einer Senke landen. Danach geht man bis zum letzten akzeptierenden Zustand zurück und gibt den bis dort hin gelesenen String aus.

3.2 Abstract Syntax Tree