Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

gedcom7.js

Realisierung einer JavaScript-Bibliothek für das genealogische Austauschformat FamilySearch GEDCOM Version 7

Marius Müller & David Gruber

Bachelor-Projektarbeit

Betreuer: Christian Bettinger

Trier, 28.02.2023

Kurzfassung

In der Kurzfassung soll in kurzer und prägnanter Weise der wesentliche Inhalt der Arbeit beschrieben werden. Dazu zählen vor allem eine kurze Aufgabenbeschreibung, der Lösungsansatz sowie die wesentlichen Ergebnisse der Arbeit. Ein häufiger Fehler für die Kurzfassung ist, dass lediglich die Aufgabenbeschreibung (d.h. das Problem) in Kurzform vorgelegt wird. Die Kurzfassung soll aber die gesamte Arbeit widerspiegeln. Deshalb sind vor allem die erzielten Ergebnisse darzustellen. Die Kurzfassung soll etwa eine halbe bis ganze DIN-A4-Seite umfassen.

Hinweis: Schreiben Sie die Kurzfassung am Ende der Arbeit, denn eventuell ist Ihnen beim Schreiben erst vollends klar geworden, was das Wesentliche der Arbeit ist bzw. welche Schwerpunkte Sie bei der Arbeit gesetzt haben. Andernfalls laufen Sie Gefahr, dass die Kurzfassung nicht zum Rest der Arbeit passt.

Abstract

The same in English.

Inhaltsverzeichnis

Abbildungsverzeichnis

1

Einleitung und Problemstellung

In dieser Ausarbeitung ...

1.1 Anforderungsanalyse & Ziele

Folgende Anforderungen werden an die Bibliothek gestellt:

- AF01: Dateien oder Strings im Format Gedcom7 sollen eingelesen werden können
- AF02: Dateien sollen im Gedcom7 Format ausgegeben werden können
- AF03: Die Syntax von Dateien oder Strings soll gemäß der Gedcom7-Spezifikation überprüfbar sein
- AF04: Die in der Gedcom7-Spezifikation definierten Datentypen sollen unterstützt werden
- AF05: Eingelesene Dateien sollen gemäß der Gedcom7-Spezifikation verändert und erweitert werden können
- AF06: Die Bibliothek soll erweiterbar sein

2

Theoretische Grundlagen

In diesem Kapitel...

2.1 Genealogie und FamilySearch

Genealogie ist ein Überbegriff für die Familien- und Ahnenforschung und beschäftigt sich mit der historischen Herkunft und der Geschichte von Menschen weltweit [Ahn]. Dabei sind insbesondere Abstammungs- und Verwandtschaftsverhältnisse von besonderer Bedeutung, die anhand von Beiweisen aus validen Quellen in Stammbäumen zusammengefasst werden, die aufzeigen, wie eine Generation mit der nächsten verbunden ist. Auf Basis der so erlangten Erkenntnisse kann eine Familiengeschichte erstellt werden, die eine biographische Studie einer genealogisch nachgewiesenen Familie und der Gemeinde in der sie lebten, darstellt [Gen].

Das Aufkommen des Internets stellte einen Wendepunkt in der Genealogie dar.

2.2 GEDCOM Version 7

Das Datenformat FamilySearch GEDCOM 7.0 wurde 2021 von der Kirche Jesu Christi der Heiligen der Letzten Tage entwickelt und stellt ein einheitliches, flexibles Format für den Austausch von genealogischen Daten bereit. Das Ziel besteht darin, eine langfristige Speicherung von genealogischen Informationen zu ermöglichen, die für zukünftige Genealogen und die von ihnen verwendeten System zugänglich und verständlich ist [Fam22]. Die im Rahmen dieser Arbeit verwendete Version 7.0.11 wurde am 01.11.2022 veröffentlicht und stellt die aktuelleste[1] Version des Standards dar.

GEDCOM ist ein UTF-8 kodiertes hierarchisches Containerformat, das die Dateinamenserweiterung *.ged* verwendet. Der erste Character einer GEDCOM-Datei sollte das Byte-Order-Mark (U+FEFF) sein. Der Inhalt einer GEDCOM-Datei ist in sog. *Structures* unterteilt, die aus einem *Structure Type* und einem optionalen *Payload* bestehen und mehrere Substrukturen besitzen können. Hat eine *Structure* eine *Substructure*, dann ist die *Structure* die *Superstructure* der *Structure*. Jede

[1] Stand 31.01.2023

Substructure hat genau eine *Superstructure* und ist so in der Gesamtstruktur eindeutig zugeordnet. Eine *Structure*, die keine *Supertructure* besitzt, heißt *Record*. Alle Records zusammen mit einer *Header*- und einer *Trailer*-Struktur bilden ein *Dataset*, das den Inhalt einer GEDCOM-Datei darstellt. [Fam22]

Der *Payload* einer *Structure* ist eine Zeichenkette eines bestimmten Datentyps, die entweder Informationen für die *Supertructure* bereithält, oder einen Zeiger auf eine andere *Structure* repräsentiert und somit auf diese verweist. GEDCOM v7 definiert 11 verschiedene Datentypen in [Fam22] mit denen Namen, Daten, Uhrzeiten, Texte und vieles mehr dargestellt werden können. Der *Structure Type* ist eindeutig definiert durch eine URI und gibt an, welche Bedeutung und welchen Datentyp die *Structure* besitzt, welche *Substructures* enthalten sein können und mit welcher Kardinalität diese auftreten können. [Fam22]

Kodiert wird der Inhalt einer GEDCOM-Datei in sog. *Lines*, die eine Zeichenkettenrepräsentation einer Struktur (bzw. eines Teils einer Struktur) darstellen und wie folgt aufgebaut sind (eckige Klammern repräsentieren optionale Inhalte):

Level D [Xref D] Tag [D LineVal] EOL

- Level: Eine Line beginnt mit einem Level, das die Verhältnisse der *Structures* untereinander beschreibt. Alle *Structures* mit dem kleinstmöglichen Level 0 sind Records - Level ≥ 1 repräsentieren *Substructures*. Eine *Structure* mit dem Level x ist also die *Superstructure* aller folgenden *Structures* mit dem Level $x + 1$.
- D: *D* steht für *Delimiter*, was englisch für Trennzeichen ist und repräsentiert in diesem Fall das Leerzeichen mit dem Unicode $u + 0020$.
- Xref: Xref ist die Abkürzung für *Cross-Reference Identifier* und fungiert als Adresse für eine *Structure*. Möchte man von einer *Structure* auf eine andere *Structure* verweisen, kann dies über einen Zeiger-Payload auf die entsprechende *Structure* realisiert werden.
- Tag: Der *Tag* kodiert den *Structure Type* einer *Structure*.
- LineVal: Im *LineVal* einer Struktur ist der Payload kodiert.
- EOL: EOL steht für End-Of-Line und kodiert das Ende einer Line. Im Format GEDCOM v7 kann dies entweder durch einen Carriage-Return (Unicode U+000D), Line-Feed(Unicode U+000A) oder einen Carriage-Return gefolgt von einem Line-Feed repräsentiert werden.

Ein Ausschnitt aus einer GEDCOM-Datei ist in Listing 2.1 dargestellt. Dieser Ausschnitt zeigt einen *Record* vom Typ *Individual*, in dem Informationen über ein Individuum gespeichert werden können. Dem Individuum Cross-Reference Identifier *@I1@* zugewiesen, sodass im Dokument auf dieses verwiesen werden kann. In diesem Fall handelt es sich um ein männliches Individuum mit dem Namen John Doe. Über die *Structure* mit dem *Tag BIRT* kann das Geburtsdatum (Birthdate) angegeben werden, das in diesem Fall auf den 1.März 1951 datiert ist. Mit der *Structure FAMS* wird eine Zugehörigkeit zur Family mit dem Cross-Reference Identifier *@F2@* ausgedrückt.

```
0 @I1@ INDI
1 NAME John Doe
1 SEX M
1 BIRT
2 DATE 1 MAR 1951
1 FAMS @F2@
```

Listing 2.1: Beispiel für einen Individual Record

Detaillierte Erklärungen, alle Informationen zu *Structure Types*, Datentypen, usw. und viele weitere Beispiele können in [Fam22] nachgelesen werden.

2.3 Nearley

2.4 Mocha

3

Related Work

In diesem Kapitel...

4

Konzept

Die Bibliothek *gedcom7.js* lässt sich wie in Abbildung 4.1 dargestellt in vier logische Teile gliedern. Das zentrale Element ist der Gedcom Parser, mit dem Dateien oder Strings im Format Gedcom7 eingelesen werden und mit Hilfe von *Nearley* auf Korrektheit der Syntax überprüft werden können. Die dafür zugrundeliegende Grammatik wird mit Hilfe eines *Grammatik Generators* generiert, der die in [Fam22] definierte Spezifikation in eine nearley-konforme Syntax überführt. Die so eingelesenen Informationen werden in Gedcom Datenstrukturen gespeichert, die verändert und erweitert werden und anschließend im Format Gedcom7 ausgegeben werden können. In den folgenden Abschnitten werden die vier Teile und das Zusammenspiel dieser in detaillierter Form vorgestellt.

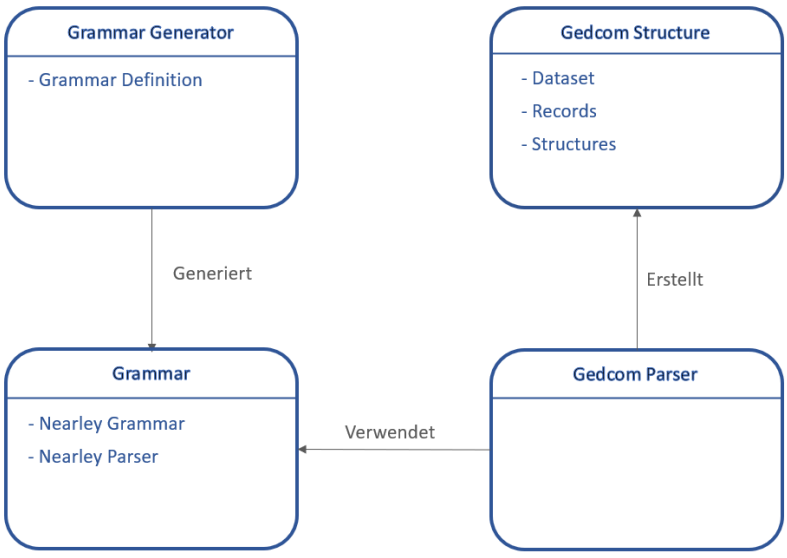

Abbildung 4.1: Allgemeiner Aufbau

4.1 Gedcom Grammatik

Eine wichtige Anforderung für die Bibliothek ist, dass die Syntax von Dateien oder Strings, die eingelesen werden, gemäß der Gedcom7-Spezifikation überprüfbar sein soll. Da die Gedcom Datenstrukturen veränderbar und erweiterbar sein sollen, ist es wichtig, dass eine Syntaxüberprüfung nach Änderungen auf einfache Weise möglich ist. Umgesetzt wird diese Syntaxüberprüfung mit Hilfe des in Kapitel 2.3 vorgestellten JavaScript-Parser-Toolkits *Nearley*. Mit *Nearley* können auf einfach Weise, menschenlesbare Grammatiken erstellt und zu einem *Nearley-Parser* kompiliert werden. Von großem Vorteil ist dabei, dass Features wie Postprozessoren und die Implementierung eines Lexers unterstützt werden.

4.1.1 Pre- und Postprozessor

Standardmäßig packt ein *Nearley-Parser* jedes Zeichen, das mit einer Regel übereinstimmt, in ein Array [Cha]. Bei komplexeren Grammatiken wie der Gedcom7 Spezifikation führt dies dazu, dass sehr viele Arrays innereinander verschachtelt werden, sodass schnell zweistellige Verschatlungsgrade erreicht werden, was ein weiterarbeiten mit den Ergebnissen erschwert. Mit Hilfe von Postprozessoren können jeder *Nearley Regel* Verarbeitungsanweisungen zugewiesen werden, sodass die Ergebnisse beispielsweise im JSON-Format zurückgegeben werden. Auf diese Weise können die eingelesenen Dateien bereits bei der Syntaxüberprüfung in eine passende Darstellungsform gebracht werden, sodass eine leichte Überführung in die passende Gedcom Datenstruktur möglich ist.

Desweiteren kann ein Lexer verwendet werden, um die Arbeit mit *Nearley* zu optimieren. Ein *Nearley-Parser* teilt die Eingabedaten standardmäßig in einen Strom von einzelnen Zeichen, die sequentiell abgearbeitet werden, was auch als *Scannerless Parsing* bezeichnet wird. Ein Lexer ist eine Art Preprozessor, der die Eingabedaten in größere Einheiten, die sog. *Tokens* zusammenfasst [Cha]. Auf diese Weise wird der Aufwand beim Parsen verringert und die Interpretation der Eingabedaten fällt oft leichter. Ein einfaches Beispiel hierfür ist eine Regel die einen Zahlenwert erwartet. Ist der Eingabewert beispielsweise "137", würde ein *Nearley-Parser* standardmäßig jede Ziffer einzeln einlesen und im Postprozessor müsste definiert werden, dass die aufeinanderfolgenden Ziffern als ein Zahlenwert interpretiert werden sollen. Mit Hilfe eines Lexers könnte eine einfache Regel definiert werden, die den kompletten Zahlenwert als ein Token vorverarbeitet. Im Rahmen dieser Arbeit wurde der JavaScript Lexer *Moo.js* [Rad] verwendet. *Moo.js* zeichnet sich durch seine Geschwindigkeit[1] aus und wird von *Nearley* als Lexer unterstützt.

4.1.2 Nearley-Parser für Gedcom7

Da die Gedcom7- sowie die Nearley Syntax beide auf EBNF-Sprachkonzepten basieren, lässt sich die Gedcom7 Spezifikation ohne weiteres in eine Nearley Grammatik übersetzten, die dann zu einem Nearley-Parser kompiliert werden kann. Wird

[1] Laut den Entwicklern ist Moo.js der schnellste JavaScript-Lexer und ∼2-10 mal schneller als herkömmliche Lexer [Rad].

diesem Nearley-Parser eine Gedcom7-Datei (.ged) kodiert als UTF-8 Zeichenkette übergeben, erfüllt dieser die folgenden zwei Aufgaben:

1. Überprüfung der Gedcom7 Syntax

Der Nearley-Parser überprüft den übergebenen Gedcom7-String Line für Line, indem er alle Zeichen (bzw. Tokens) sequentiell liest, bis ein End-Of-Line (EOL) Token gefunden wird. Nach jedem Zeichen das eingelesen wird, überprüft der Parser, welche in der Grammatik definierten Regeln durch das neu eingelesene Zeichen nicht mehr mit der Zeichenkette übereinstimmen und verwirft diese. Wird ein EOL Token gelesen werden die Postprozessoren aller übereinstimmenden Regeln ausgeführt und ein Array mit den Ergebnissen dieser Postprozessoraufrufe als Ergebnis der Line zurückgegeben. Da die Gedcom7 Grammatik nicht mehrdeutig ist, findet der Parser bei korrekter Gedcom7 Syntax immer ein eindeutiges Ergebnis[2] (d.h. beim Erreichen des EOL Tokens ist maximal eine übereinstimmende Regel übrig). Werden bei diesem Prozess alle Regeln der Grammatik ausgeschlossen, bevor ein EOL Token gelesen wird, ist die Syntax des übergebenen Gedcom7-Strings nicht korrekt und ein Syntaxfehler kann erzeugt werden. Da die Zeichenkette sequentiell abgearbeitet wird, kann bei auftretendem Fehler genau aufgezeigt werden, welche Line und welches Zeichen fehlerhaft sind.

2. Extrahieren der Strukturinformationen

Eine weiterer Aufgabe des Nearley-Parsers ist es, die Strukturinformationen des Gedcom7-Strings zu extrahieren, sodass im nächsten Schritt eine einfache Überführung in entsprechende Gedcom Datenstrukturen möglich ist. Durch den sequentiellen Aufbau einer Gedcom7-Datei wird eine Struktur stets vor seinen Substrukturen definiert. Da das erste Token jeder Line stets das Level der Line repräsentiert, kann der Nearley-Parser die Abhängigkeiten der Lines zueinander zuordnen und es ist zu jedem Zeitpunkt eindeutig, welcher Superstruktur eine Struktur zugeordnet werden soll. Folgende Informationen können also durch den Nearley-Parser extrahiert werden:

- **URI**: Auch wenn bestimtme Structuretypes denselben Tag besitzen, kann aus der Kombination von Level und Tag die eindeutige Gedcom URI bestimmt werden
- **Datentyp**: Sofern ein Payload in der Line vorhanden ist, kann mit Hilfe der URI der Datentyp des Payloads bestimmt werden
- **Superstruktur**: Zu jeder Line kann die entsprechende Superstruktur angegeben werden, sofern es sich nicht um einen Record (Structure mit Level 0) handelt, die keine Superstructure besitzen
- **Substrukturen**: Hat eine Sturktur eine oder mehrere Substrukturen, können diese auf Basis des Levels der Lines bestimmt werden

[2] Hier Kapitel ansprechen in dem über ambigious grammar geredet wird, bei level problem

4.2 Grammatik Generator

In der Gedcom7 Spezifikation werden 181 Structuretypes verteilt auf 8 Records definiert, die alle in einer Line der Form

Level D [Xref D] Tag [D LineVal] EOL

dargestellt werden. Sollen diese Structuretypes in eine Nearley Grammatik überführt werden, muss für jede dieser Strukturen und jede mögliche Kombination an Substrukturen eine Regel erstellt werden. Da dies eine sehr repetitive Aufgabe ist und sich die Regeln nur an bestimmten Stellen unterscheiden, lässt sich die Grammatikerstellung durch einen Grammatik Generator automatisieren. Dazu können Definitionsdateien erstellt werden, die die für alle Structuretypes die folgenden Informationen bereithalten:

- **URI**: Die URI des Structuretypes wird benötigt, um eine Struktur eindeutig zuordnen zu können
- **LineType**: Der LineType gibt an, wie die Line aufgebaut ist (Cross-Reference-Identifier vorhanden? Payload vorhanden?)
- **Datatype**: Sofern ein Payload in der Line vorhanden ist, kann über den Datatype die Syntax des Payloads ermittelt werden
- **Tag**: Der Tag wird benötigt, damit die Nearley Regeln eindeutig sind
- **Substructures**: In der Gedcom7 Spezifikation sind für alle Structuretypes alle möglichen Substructures definiert. Mit dieser Information können alle korrekten Fälle in Nearley Regeln abgebildet werden
- **Level**: Um eine eindeutige Grammatik zu generieren, müssen die Level mit denen ein jeweiliger Structuretype auftreten kann, zwingend mit angegeben werden. Da in der gedcom7 Spezifikation TAGs mehrfach für verschiedene Typen verwendet werden, kann nicht einfach ein generisches Level für die Regeln verwendet werden, das ganzzahlige Werte akzeptiert, da die entstehende Grammatik damit mehrdeutig wäre. Ein Beispiel hierfür sind die Structures *g7:HEAD-DATE* und *g7:DATE-exact* im Header Record. Mit einem generischen Level wären die Regeln für beide Structuretypes identisch mit

  Level D "DATE"D DateExact EOL

  Wird eine solche Line als Substructure eines Header Records von dem Nearley Parser gelesen, kann dieser nicht entscheiden, ob es sich um ein *g7:HEAD-DATE* oder ein *g7:DATE-exact* handelt und würde somit zwei Ergebnisse aufrecht erhalten. Um diese Mehrdeutigkeit zu verhindern, wird das Level in der Definition angegeben.

Anhand dieser Informationen kann der Grammatik Generator automatisiert Nearley Regeln formulieren. Diese Regeln können zu einer Grammatik zusammengefasst und anschließend vom Generator zu einem Nearley-Parser kompiliert werden. Anhand des LineTypes kann der Generator den Regeln die passenden Postprozessoren zuweisen, die für das Extrahieren der Strukturinformationen zuständig sind. Auf diese Weise kann ein voll funktionaler Nearley-Parser automatisiert generiert

werden, der die Gedcom7-Syntax vollständig parsen und alle für die weitere Verarbeitung benötigten Informationen extrahieren kann.

Ein weitere großer Vorteil an dieser Automatisierung ist, dass zur Erfüllung der Anforderung der einfachen Erweiterbarkeit der Bibliothek beigetragen wird. Sollte die Bibliothek in zukünftigen Projekten durch neue Strucutretypes o.ä. erweitert werden, ist dies auf einfache und verständliche Weise durch das Hinzufügen neuer Einträge in die Strukturdefinitionen möglich. Desweiteren bildet der Grammatik Generator ein Fundament für einen wichtigen Use-Case, der in weiterführenden Arbeiten adressiert werden sollte: der Möglichkeit Extensions zu definieren. Die Gedcom7 Spezifikation definiert die wichtigsten Strukturen zur Speicherung genealogischer Informationen - für alle Informationen die über diese Standardstrukturen hinausgehen, müssen Extensions definiert werden. Da genealogische Informationen sehr vielfältig sein können, sind Extensions ein probates Mittel, dass in vielen Anwendungen genutzt wird. Mit Hilfe des Grammatik Generators kann die Definition von Extensions umgesetzt werden, indem eine Schnittstelle zum Generator entwickelt wird, die dem Benutzer zur Verfügung gestellt wird. Über diese Schnittstelle kann die Strukturdefinition erweitert werden und anschließend die Grammatik neu generiert und kompiliert werden. Auf diese Weise könnte die Bibliothek auf die Anforderung aller Benutzer angepasst werden.

4.3 Gedcom Strukturen

4.4 Gedcom Parser

5

Implementierung & Test

In diesem Kapitel...

5.1 Gedcom Grammatik

5.1.1 Gedcom7 Syntax in Nearley

Da die Gedcom7- sowie die Nearley Syntax beide auf EBNF-Sprachkonzepten basieren, lässt sich die Gedcom7 Spezifikation ohne weiteres in eine Nearley Grammatik übersetzten. Um Nearley Regeln für eine *Gedcom Line*[1] zu definieren, können die folgenden Tokens für das Leerzeichen, den *Cross-Reference Identifier* und die End-Of-Line Zeichenfolge in Form von regulären Ausdrücken definiert werden:

```
D    : /[ ]/
Xref : /\@[A-Z0-9\_]+\@/
EOL  : /(?:\r\n?|\n)/
```

Listing 5.1: Tokens für eine Gedcom Line, definiert als regulärer Ausdruck

Diese regulären Ausdrücke werden in der Vorverarbeitungsphase vom Moo-Lexer verwendet, um zusammenhängende Zeichen zu Tokens zu gruppieren, die dann in der Nearley Grammatik über den Tokennamen mit einem vorangestellten %-Zeichen angesprochen werden können. Soll nun die erste Line eines Individual-Records geparsed werden, könnte dies mit der folgenden Nearley-Regel umgesetzt werden:

```
record_Indi -> "0" %D %Xref %D "INDI" %EOL
```

Listing 5.2: Nearley Regel zum parsen eines Individual Records

[1] siehe Kapitel GEDCOM Version 7

Diese Regel würde die erste Line des in Listing XY vorgestellten Individual-Records als Eingabe akzeptieren.

5.2 Grammatik Generator

5.3 Gedcom Struktur

5.4 Gedcom Parser

6

Zusammenfassung und Ausblick

In dieser Arbeit wurde ...

Literaturverzeichnis

Ahn. Ahnenforschung: *Genealogie*. Abgerufen am 02.02.2022 von `https://www.ahnenforschung.de/themen/genealogie/`.

Cha. Chandra, Kartik: *Documentation: nearley.js*. Abgerufen am 10.02.2022 von `https://nearley.js.org/`.

Fam22. Family History Department: *The FamilySearch GEDCOM Specification 7.0.11*. The Church of Jesus Christ of Latter-day Saints, 15 East South Temple StreetSalt Lake City, UT 84150 US, 7.0.11 Auflage, November 2022.

Gen. Genealogists, Society Of: *Genealogy or Family History*. Abgerufen am 02.02.2022 von `https://www.sog.org.uk/learn/hints-tips/genealogy-or-family-history`.

Rad. Radvan, Tim: *Documentation: moo.js*. Abgerufen am 10.02.2022 von `https://github.com/no-context/moo`.

A

Glossar

GEDCOM	GEnealogical Data COMunication
URI	Uniform Resource Identifier

B

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Seminararbeit ohne fremde Hilfe verfasst und nur die im Literaturverzeichnis angegebenen Quellen verwendet habe.

Datum	Unterschrift der Kandidatin/des Kandidaten