UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Abbilden und Nutzen von versionsübergreifenden medizinischen Klassifikationen mittels FHIR ConceptMaps

Visualization and Mapping of Medical Classifications across Multiple Versions with FHIR ConceptMaps

Masterarbeit

verfasst am
Institut für Medizinische Informatik

im Rahmen des Studiengangs
Medizinische Informatik
der Universität zu Lübeck

vorgelegt von
Simon Müller

ausgegeben und betreut von
Prof. Dr. Josef Ingenerf

mit Unterstützung von
Tessa Ohlsen, M. Sc.

Lübeck, den 1. November 2024

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____

Simon Müller

Zusammenfassung

Abstract

Inhaltsverzeichnis

1

Einleitung

Medizininformatik-Initiative `https://www.medizininformatik-initiative.de/de/start`

Deutsches Forschungsdatenportal für Gesundheit `https://forschen-fuer-gesundheit.de`

1.1 Grundbegriffe

Klassifikationen ICD-10-GM und OPS

FHIR ConceptMap

«HL7® FHIR® (im Folgenden „FHIR“ genannt) ist der modernste Interoperabilitäts-Standard aus der Produktfamilie von Health Level 7 International (kurz: „HL7“), einer internationalen Standardisierungsorganisation für das Gesundheitswesen, die in der Vergangenheit schon viele erfolgreiche und weit verbreitet genutzte Standards, wie zum Beispiel HL7 Version 2 oder HL7 CDA (Clinical Document Architecture) hervorgebracht hat. [...] HL7 wurde 1987 gegründet, um Standards für klinische Informationssysteme zu erarbeiten. [...] FHIR ist die dritte Generation von Interoperabilitätsstandards aus der Feder von HL7. Die Entwicklung begann im Jahre 2011 als Reaktion auf die Forderungen aus der Industrie nach einer standardisierten Lösung für die Entwicklung webbasierter Applikationen für das Gesundheitswesen.» (Heckmann, 2022)

„Health Level 7 wurde 1987 gegründet, um Standards für klinische Informationssysteme zu erarbeiten. [...] FHIR ist die dritte Generation von Interoperabilitätsstandards aus der Feder von HLR7. Die Entwicklung begann im Jahre 2011 als Reaktion auf die Forderungen aus der Industrie nach einer standardistieren Lösung für die Entwicklung webbasierter Applikationen für das Gesundheitswesen.“ (Heckmann, 2022, Seite 309)

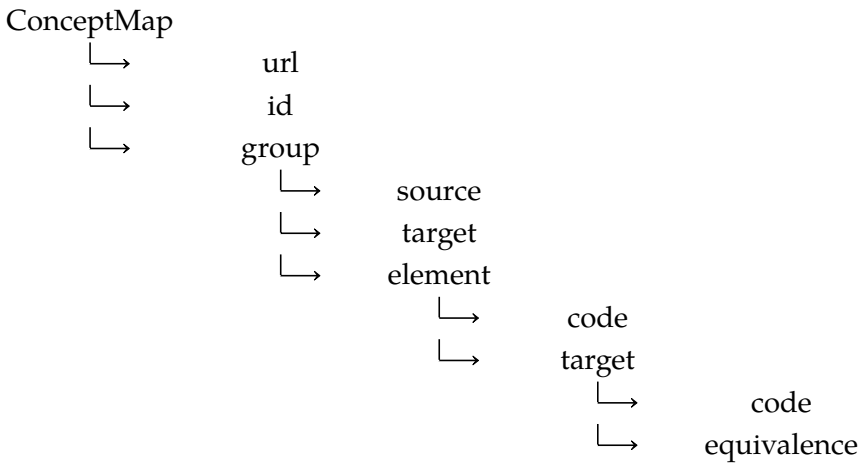
FHIR (Braunstein, 2022)

REST

REST (Fielding, 2000)

ConceptMap

http://hl7.org/fhir/R4/conceptmap.html



- ConceptMap: A statement of relationships from one set of concepts to one or more other concepts - either concepts in code systems, or data element/data element concepts, or classes in class models.
  A map from one set of concepts to one or more other concepts
- url: An absolute URI that is used to identify this concept map when it is referenced in a specification, model, design or an instance; also called its canonical identifier. This SHOULD be globally unique and SHOULD be a literal address at which at which an authoritative instance of this concept map is (or will be) published. This URL can be the target of a canonical reference. It SHALL remain the same when the concept map is stored on different servers.
  Canonical identifier for this concept map, represented as a URI (globally unique)
  § wird von HAPI benötigt
- id: The logical id of the resource, as used in the URL for the resource. Once assigned, this value never changes.
  Logical id of this artifact § kann nicht verändert werden
- group: A group of mappings that all have the same source and target system.
  Same source and target systems
- source: An absolute URI that identifies the source system where the concepts to be mapped are defined.
  Source system where concepts to be mapped are defined
- target: An absolute URI that identifies the target system that the concepts will be mapped to.
  Target system that the concepts are to be mapped to
- element: Mappings for an individual concept in the source to one or more concepts in the target.
  Mappings for a concept from the source set

- code: Identity (code or path) or the element/item being mapped.
  Identifies element being mapped
- target: A concept from the target value set that this concept maps to.
  Concept in target system for element
- code: Identity (code or path) or the element/item that the map refers to.
  Code that identifies the target element
- equivalence: The equivalence between the source and target concepts (counting for the dependencies and products). The equivalence is read from target to source (e.g. the target is 'wider' than the source).
  § Auflisten, alle Versionen
- unmapped: What to do when there is no mapping for the source concept. Ünmapped"does not include codes that are unmatched, and the unmapped element is ignored in a code is specified to have equivalence = unmatched.
  What to do when there is no mapping for the source concept
  § provided

1.2 Verwandte Arbeiten

Normen für Mappings

Health Informatics, Terminology resource map quality measures (MapQual) (ISO 21564, 2019)

Health informatics – Principles of mapping between terminological resources (ISO 12300, 2014)

Medicats

ConceptMap für UMLS

using multiple groups (Saripalle, 2019)

Interlingua SNOMED CT

SNOMED (Philipp u. a., 2022)

1.3 Aufbau & Beiträge dieser Arbeit

2

BfArM-Daten

Dieses Kapitel behandelt die elektronische Verarbeitung der BfArM-Daten.

2.1 Kode- und Umsteiger-Dateien

Das BfArM stellt für jede neue ICD-10-GM- und OPS-Version Dateien für die Überleitung auf die Vorgänger-Version zum Download zur Verfügung, gebündelt jeweils in einer Zip-Datei: (**bfarmdl**).

Es handelt sich hierbei um CSV-formatierte Text-Dateien. CSV steht für „Comma-Separated Values“ und ist ein sehr einfaches Format, um Daten zu strukturieren. Es wird ein Satzzeichen verwendet, um den restlichen Text in Spalten zu trennen – laut dem Namen normalerweise ein Komma, aber für die BfArM-Dateien wurde Strichpunkt als Trennzeichen gewählt, wahrscheinlich weil die Klassentitel auch Kommata enthalten können. Weitere Informationen zum CSV Datei-Format finden sich hier: (**bonnefoy2024definitive**).

Vom BfArM werden Kodes als Schlüsselnummern bezeichnet, wenn diese eindeutig sind und einzelne Überleitungen zwischen Kodes werden Umsteiger genannt.

Kodes / Schlüsselnummern

Hier beispielhaft die ersten sieben Zeilen der Kode-Datei von ICD-10-GM, Version 2024:

```
UNDEF;Undefined
A00;Cholera
A00.0;Cholera durch Vibrio cholerae O:1, Biovar
cholerae
A00.1;Cholera durch Vibrio cholerae O:1, Biovar eltor
A00.9;Cholera, nicht näher bezeichnet
A01;Typhus abdominalis und Paratyphus
A01.0;Typhus abdominalis
```

Anmerkungen:

- Ein Strichpunkt = zwei Spalten
  1. Kode
  2. Klassentitel
- Für OPS ist das Format der Kode-Datei identisch.
- Mit Ausnahme des UNDEF-Eintrags in der ersten Zeile ist die Datei alphabetisch nach dem Kode sortiert. UNDEF ist kein ICD-10-GM- oder OPS-Kode, sondern wird für Umsteiger verwendet, um entfernte beziehungsweise neu hinzugefügte Kodes zu kennzeichnen.
- Die Datei enthält nicht-endständige Kodes – im Beispiel oben A00 und A01. Ein Kode ist endständig, wenn er keine Subkategorie hat. (**bfarmicdkk**)

Umsteiger / Überleitungen

Im Gegensatz zu den Kodes haben die Umsteiger für ICD-10-GM und OPS unterschiedliche Formate. Hier also zuerst zwei Ausschnitte aus der Umsteiger-Datei für ICD-10-GM, Version 2017 Überleitung auf Version 2016:

```
A00.0;A00.0;A;A
A00.1;A00.1;A;A
A00.9;A00.9;A;A
A01.0;A01.0;A;A
A01.1;A01.1;A;A
--------------------
U06.0;UNDEF;A;
UNDEF;Z99.0;;
Z99.0;Z99.0;A;
```

Anmerkungen:

- Drei Strichpunkte = vier Spalten
  1. Alter Kode (2016)
  2. Neuer Kode (2017)
  3. Wenn A: automatisch überleitbar von 2016 auf 2017, sonst nicht
  4. Wenn A: automatisch überleitbar von 2017 auf 2016, sonst nicht
- Der obere Abschnitt umfasst die fünf ersten Zeilen der Umsteiger-Datei.
- Der untere Abschnitt enthält beispielhaft zwei Umsteiger mit UNDEF. UNDEF als neuer Kode heißt der alte Kode wurde entfernt. UNDEF als alter Kode heißt der neue Kode wurde hinzugefügt. In diesem Beispiel wurde Z99.0 umbenannt.

- Die Datei ist alphabetisch nach dem alten Kode sortiert und falls dieser bei mehreren Einträgen identisch ist, anschließend nach dem neuen Kode.
- Es sind nur endständige Kodes enthalten.

Dazu im Vergleich ein einzelner Umsteiger aus dem OPS, Version 2024 Überleitung auf Version 2023:

```
1-100;N;1-
100;N;A;A
```

Die zusätzlichen Spalten jeweils nach den Kodes speziell für OPS sagen aus, ob Zusatzkennzeichen notwendig sind, siehe dazu auch: (**bfarmopskk**).

„DRY"-Prinzip

„Don't Repeat Yourself" ist eines der Kardinalprinzipien in der Software-Entwicklung. Obwohl der Grundsatz, Wiederholungen zu vermeiden, wahrscheinlich schon in der Programmierung angewandt wird seit es diesen Beruf gibt, wurde „DRY" erstmals 1999 ausformuliert von (**thomas2019pragmatic**). In der zwanzigjährigen Jubiläumsausgabe verdeutlichen die Autoren, dass es ihnen hierbei nicht nur um das Schreiben von Programmcode geht, sondern vielmehr um die Absichten hinter einem Prozess. Das heißt eine Änderung der Intention einer Software-Lösung sollte nicht mehrere Änderungen an mehreren Stellen nach sich ziehen.

Bei der Integration der BfArM-Daten kann das „DRY"-Prinzip auf zwei Arten angewandt werden.

1. Bezogen auf Kodiersysteme: Alle Funktionen sollten unabhängig davon funktionieren, ob es sich um ICD-10-GM- oder OPS-Daten handelt. Auch die Aufnahme eines zusätzlichen Systems, beispielsweise ATC, sollte möglichst nur Anpassungen erfordern, die durch Abweichungen in der Integration der Daten dieses Systems notwendig sind.
2. Bezogen auf Versionen: Der Prozess der Datenintegration sollte unabhängig von der Version gleich ablaufen und das gleiche Ergebnis liefern, bezogen auf die Datenstruktur. Jede Abweichung zwischen Versionen sollte nur eine möglichst einfach zu implementierende Modifikation des Gesamtprozesses darstellen. Konkret heißt das beim Hinzufügen einer neuen Version, dass an nur einer Stelle die Abweichungen von der Standardversion angegeben werden sollten und der Datenintegrationsprozess danach einmal angestoßen wird. Idealerweise ändert sich bei einer neuen Version nur die Versionsnummer und die Download-URL.

Standardverfahren und Abweichungen

Im diesem Abschnitt werden alle Abweichungen der ICD-10-GM- und OPS-Versionen von der als Standard gewählten Version 2024 in Tabellen aufgeführt. Konkret gemeinst sind damit: Version, Download-URL, Pfad der Kode- und Umsteiger-Dateien, Sonstiges.

Diese Informationen können dann dem Datenintegrationsprozess in einem strukturierten Dateiformat zur Verfügung gestellt werden. §TODO: Verweis auf konkrete Implementation

Liste aller Versionen

Im Anhang A.1 befinden sich Tabellen, die alle Abweichungen zwischen den Versionen für ICD-10-GM und OPS bezogen auf 2024 enthalten.

Datei-Adressen und -Pfade

Die Download-URL der Zip-Dateien setzt sich wie folgt zusammen:

`https://multimedia.gsb.bund.de/BfArM/downloads/klassifikationen/` …
& für ICD-10-GM: `icd-10/` …
& für OPS: `ops/` …
& einen pro Version unterschiedlichen Teil, siehe URL-Eintrag in den Tabellen.
Die URL dient damit als „Single Source of Truth“ (**bonnefoy2024definitive**).

Die Kode- und Umsteiger-Dateien sind in einem Verzeichnis enthalten:

`Klassifikationsdateien`
Wenn die Tabellen einen Verzeichnis-Eintrag enthalten, wird dieser vorangestellt.
Zum Beispiel für ICD-10-GM Version 2021:
`icd10gm2021syst-ueberl-20201111/Klassifikationsdateien`

Der Pfad der Kode-Datei lautet:

Verzeichnis …
& für ICD-10-GM: `icd10gm` …
& für OPS: `ops` …
& die Version …
& `syst.txt`
Also zum Beispiel: `Klassifikationsdateien/icd10gm2024syst.txt`
Wenn die Tabellen einen Kodes-Eintrag enthalten, gilt dieser stattdessen.

Die Umsteiger-Datei funktioniert ähnlich, nur dass der Dateiname normalerweise so ist:

für ICD-10-GM: `icd10gm` …
für OPS: `ops` …
& Version …
& `syst_umsteiger_` …
& Vorgänger-Version …
& `_` …
& Version …
& `.txt` Zum Beispiel: `ops2024syst_umsteiger_2023_2024.txt`

Sonstige Abweichungen

- Vorab-Version
  Diese Version hat noch keine Seite für die Kode-Suche. §TODO: Verweis auf Kode-Suche (Frontend)
- Zip-Unterdatei
  Die Zip-Datei der 2022 Versionen enthielt weitere Zip-Dateien. Vorher wurden alle Dateien zu einer Versionen nach Verwendungszweck nur in Unterverzeichnisse gegliedert, weswegen die gebündelte Zip-Datei insgesamt relativ groß wurde. Ab 2023 werden die Zip-Unterdateien separat zum Download angeboten.
- ISO-8859-1
  Vor 2009 waren Dateien in ISO-8859-1 kodiert, auch Latin-1 genannt, statt UTF-8. Mehr dazu unter Abschnitt 3.
- Punkt-Strich-Notation, Kreuz-Stern-System
  Die Kodes älterer ICD-10-GM Versionen hatten Sonderzeichen gemäß (**bfarmicdkk**).
- 6-Spalten-Umsteiger
  Umsteiger älterer ICD-10-GM Versionen enthielten Informationen zur Mehrfachkodierung.
- Nicht endständige Umsteiger
  Im Gegensatz zu allen anderen Überleitungen sind die Umsteiger-Einträge für ICD-10-GM 2.0 auf 1.3. auch für nicht-endständige Kodes enthalten.
- None statt UNDEF
  Von OPS Version 2009 bis 2004 wurde statt UNDEF der Bezeichner „None“ verwendet.
- KOMBI-Kode
  OPS Versionen 2.1 und 2.0 enthalten in der Kodes-Datei einen zusätzlichen Eintrag: KOMBI, „Kombinationsschlüsselnummer erforderlich“.
- 6-Spalten-Umsteiger (altes Format), 5-Spalten-Umsteiger, 4-Spalten-Umsteiger
  Die Umsteiger der OPS-Versionen von 2009 bis 2005 waren anders formatiert, weil 2005 die Informationen bezüglich Zusatzkennzeichen hinzukamen und bis 2009 die Spalten unterschiedlich angeordnet waren als in allen neueren OPS Versionen.
- 6-Spalten-Umsteiger (ursprüngliches Format)
  Die Umsteiger für OPS Version 2.1 enthielen zusätzliche Spalten wegen Mehrfachverschlüsselung wie die älteren ICD-10-GM Versionen.
- 3-Spalten-Umsteiger
  OPS Version 2.0 zeigte mit nur einer Spalte an, ob automatische Überleitungen möglich sind.
- Keine Überleitung
  Aus der ältesten Version, die Überleitungen enthält, wird zusätzlich die Kodes-Datei für die Vorgänger-Versionen verarbeitet.

2.2 Datenintegrationsprozess

Wie erwähnt durchlaufen alle Daten unabhängig von Version und Kodiersystem den gleichen Integrationsprozess. Dieser orientiert sich an dem klassischen „Extract-Transform-Load“ Modell, siehe (**bonnefoy2024definitive**).

1. *Extract:* Die Daten werden in einem bestimmten Format aus einem Quell-System extrahiert.
2. *Transform:* In einem oder mehreren Prozessen werden die Daten in ein standardisiertes Format transformiert, was zum Beispiel Bereinigung, Validierung und Imputation beinhalten kann.
3. *Insert:* Die Daten werden in ein Ziel-System integriert, um dort von weiteren Applikationen verwendet zu werden.

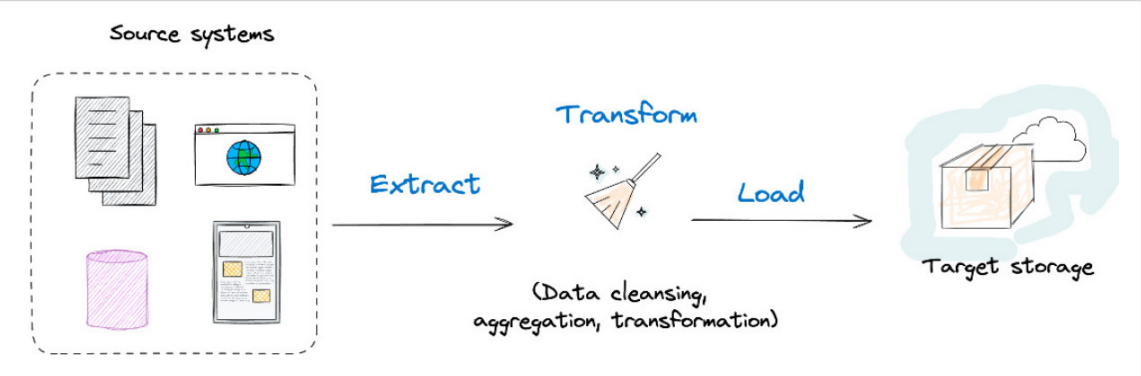


Abbildung 2.1: ETL-Modell nach (**bonnefoy2024definitive**)

Für die BfArM-Daten sieht der Integrationsprozess konkret so aus:

1. *Download:* Die Zip-Dateien werden heruntergeladen. Alternativ kann geprüft werden, ob die Dateien schon lokal vorhanden sind mit einem bestimmten Pfad, der sich nach Kodiersystem und Versionsnummer immer gleich zusammensetzt, zum Beispiel: `files/icd10gm2024.zip`. Die Download-Funktion sollte die Zip-Dateien ebenfalls unter diesem Pfad abspeichern, falls so gewünscht.
2. *Unzip:* Die Kodes- und Umsteiger-Dateien werden aus der Zip-Datei extrahiert. Normalerweise muss dafür nicht das ganze Archiv in temporäre Dateien entpackt werden – außer eventuell bei den Versionen 2022, weil das Extrahieren verschachtelter Zip-Dateien eher ein Nischenfall ist und nicht unbedingt standardmäßig von Programmiersprachen oder Bibliotheken unterstützt wird.
3. *Convert Encoding:* Die in ISO-8859-1 kodierten Kodes-Dateien müssen in UTF-8 umgewandelt werden. In (**charencoding**) werden die beiden Zeichenkodierungen genauer erklärt, aber für die BfArM-Daten ist eigentlich nur relevant, dass Umlaute mit unterschiedlichen Werten kodiert sind. Also würde das Einlesen eines in ISO-8859-1 kodierten Umlauts als UTF-8 ein anderes Zeichen als Resultat ergeben. Die

Umsteiger-Dateien sind davon nicht betroffen, weil in diesen keine Umlaute enthalten sind.

4. *Parse CSV:* Ein Parser wandelt eine Datei in eine Datenstruktur um; für CSV sollte jede Programmiersprache so eine Funktion standardmäßig zur Verfügung stellen. Für die BfArM-Dateien ist das Ergebnis ein zweidimensionales Array mit zwei Spalten für die Kodes, beziehungsweise drei bis sechs Spalten für die Umsteiger je nach Kodiersystem und Version.

5. *Process Data:* Aufgrund der oben erwähnten Abweichungen ist die Vorverarbeitung der Daten der komplexeste Schritt und wird im nächsten Abschnitt genauer erklärt. Außerdem müssen nicht alle Daten gespeichert werden. Vor allem in Bezug auf die Zip-Dateien ergibt das eine Reduktion der Datenmenge um etwa einen Faktor von zehn.

6. *Insert:* Die bearbeiteten Daten werden für die Verwendung durch Applikationen gespeichert. Zum Beispiel für eine relationale Datenbank werden pro Dateityp, Kodiersystem und Version eine Tabelle angelegt und die Daten in diese geschrieben. Konkret für SQL müssen außerdem die Hochkommata in den Codes-Dateien beachtet werden.

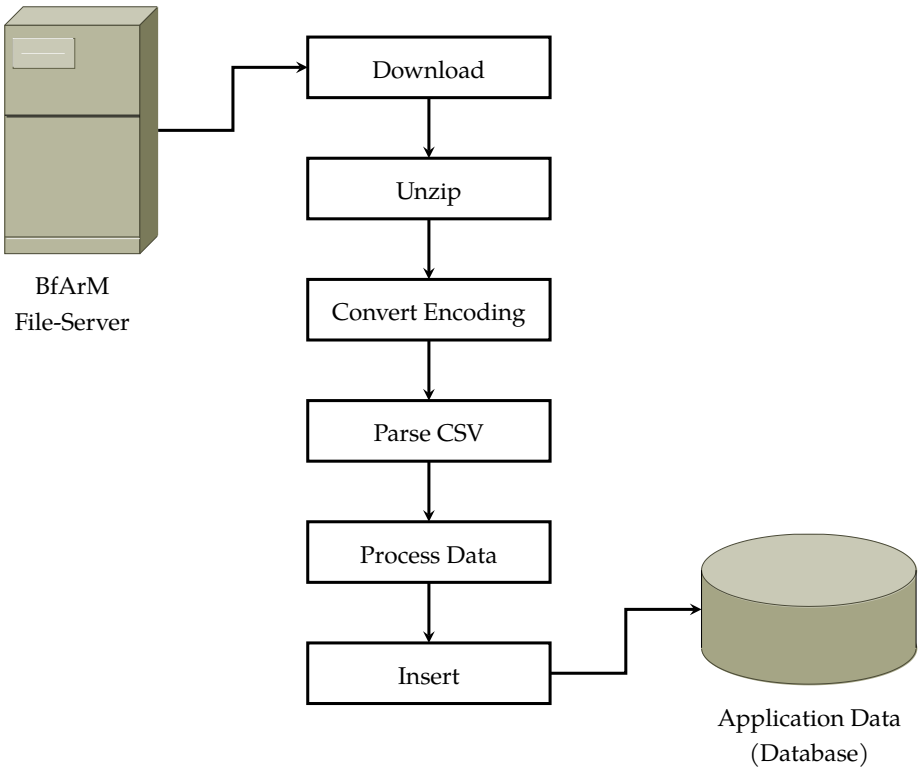


Abbildung 2.2: BfArM-Datenintegrationsprozess

2.3 Datenvorverarbeitung

„Data Preprocessing“ ist ein wichtiger Schritt in Feldern der Informatik wie *Machine Learning* und *Big Data*. In (**garcia2016big**) werden mehrere Methoden vorgestellt, wovon folgende in der Verarbeitung der BfArM-Daten zur Verwendung kommen:

1. *Data Cleaning:* Daten werden bereinigt, was sowohl das Korrigieren einzelner Werte, als auch das Entfernen überflüssiger Datensätze beinhaltet. Letzteres wird *Instance Reduction* genannt.
2. *Data Normalization:* Umwandlung der Datensätze auf ein bestimmtes Format.
3. *Data Integration:* Ein Datensatz wird durch zusätzliche Informationen bereichert, beziehungsweise mehrere Informationen werden zu einem Datensatz kombiniert.
4. *Missing values imputation:* Falls Informationen fehlen, müssen die betroffenen Datensätze mit einer bestimmten Logik behandelt werden oder alternativ können Daten durch eine Zufallsfunktion simuliert werden.

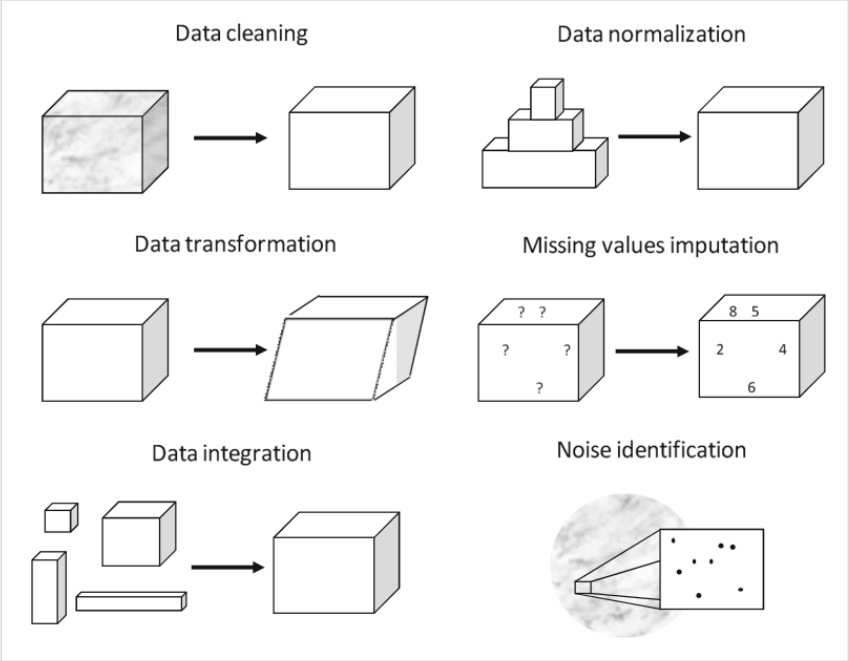


Abbildung 2.3: „Preprocessing Tasks“ aus (**garcia2016big**)

Die folgenden Unterabschnitte erklären die Vorverarbeitungsschritte für die BfArM-Daten und beziehen sich damit auf die in 2.1 genannten Abweichungen. Die Schritte erfolgen in der gelisteten Reihenfolge. Obwohl die Daten nach dem CSV-Parsing schon in einer von der Programmiersprache abhängigen Struktur vorliegen, wird zur Erklärung trotzdem noch die Datei-Struktur verwendet.

Datennormalisierung

6-Spalten-Umsteiger

Sowohl die ICD-10-GM Versionen 2004 und 2.0, als auch die OPS Version 2.1 beinhalteten Umsteiger in folgendem Format:

```
A00.0;A00.0;A;A;0;UNDEF
-------------------------
1-202;1-202;A;A;;
```

Um diese an das ICD-10-GM Format von 2024 anzupassen, werden die letzten beiden Spalten entfernt.

OPS Umsteiger

Für OPS Versionen ab 2010 sehen die Umsteiger-Einträge so aus:

```
1-100;N;1-100;N;A;A
```

Durch Entfernung der zweiten und vierten Spalte stimmen diese mit den ICD-10-GM Umsteiger Format von 2024 überein.

OPS 6-Spalten-Umsteiger, altes Format

Die OPS Versionen 2009 bis 2006 hatten ebenfalls sechs Spalten für die Umsteiger, aber in einer anderen Reihenfolge:

```
1-100;1-100;N;N;A;A
```

Hier müssen also die dritte und vierte Spalte entfernt werden.

OPS 5-Spalten-Umsteiger

Die Umsteiger von OPS Version 2005 waren in einem ganz eigenen Format geschrieben:

```
5-062.0;5-062.0;N;A;A
5-062.1;5-062.1;N;A;A
5-062.2;5-062.8;J;E;E
5-062.3;5-062.8;J;B;B
```

Hier wird dritte Spalte entfernt und außerdem werden die Sonderformen für automatische Überleitbarkeit von `B` und `E` nach `A` umbenannt.

Imputation

Für Umsteiger der OPS Version 2.0 gibt es nur drei Spalten:

```
1-208.0;A;1-209.0
-------------------------
1-208.x;;1-209.4
```

Die zweite Spalte zeigt allein die Überleitbarkeit an. Für die Angleichung an das ICD-10-GM Format von 2024 wird also die zweite Spalte entfernt und gedoppelt angehängt. Aus den beiden Beispielzeilen wird damit:

```
1-208.0;1-209.0;A;A
-------------------------
1-208.x;1-209.4;;
```

Datenbereinigung

KOMBI-Kode Aus OPS Versionen 2.1 und 2.0 wird die erste Zeile der Kode-Datei entfernt, welche den KOMBI-Eintrag enthält.

None statt UNDEF Für OPS Versionen 2009 bis 2004 wird der Kode-Wert `None` durch `UNDEF` ersetzt, sowohl in den Kodes-, als auch in den Umsteiger-Dateien.

Kreuz-Stern-System Für die ICD-10-GM Versionen 2.0 und 1.3 werden die Zeichen `+`, `*` und `!` aus den Kode-Werten entfernt – sowohl in der Kodes-, als auch der Umsteiger-Datei.

Punkt-Strich-Notation Für die ICD-10-GM Versionen 2004, 2.0 und 1.3 wird zuerst die Zeichenfolge `.-` aus den Kode-Werten in beiden Dateitypen entfernt. Danach wird nochmals `-` entfernt. Die Reihenfolge ist wichtig, weil in Version 2004 zum Beispiel Kodes `A00.-` und `G82.1-` vorkommen.

Instanzreduktion

Umsteiger

Für viele Versionen sind über 90% der Umsteiger-Einträge automatische Überleitungen in den gleichen Kode. Diese müssen also gar nicht in eine Applikation aufgenommen werden, unter der Annahme, dass nicht vorhandene Umsteiger gleich automatische Überleitungen sind. In dem Fall können alle Umsteiger ausgeschlossen werden, bei denen der neue Kode gleich dem alten Kode ist und die automatische Überleitbarkeit in beide Richtungen gegeben ist.

Nicht-endständige Umsteiger

Die Überleitung von ICD-10-GM Version 2.0 auf 1.3 ist der einzige Fall, in dem die Umsteiger-Datei nicht-endständige Kodes enthält. Durch das Entfernen der Sonderzeichen von Punkt-Strich-Notation und Kreuz-Stern-System gibt es außerdem doppelte Einträge in der er-

sten Spalte, das heißt bei den Kodes der Vorgänger-Version. Folgende Funktion[1] entfernt die überflüssigen Einträge.

Angenommen die Kodes befinden sich in einem Array mit Index von 0 bis (n-1), zum Beispiel für die ersten sechs Zeilen der Umsteiger-Datei aus ICD-10-GM, Überleitung 2.0 nach 1.3:

Index	old (Alter Kode)	new (Neuer Kode)
0	`A00`	`A00.0`
1	`A00`	`A00.1`
2	`A00`	`A00.9`
3	`A00.0`	`A00.0`
4	`A00.1`	`A00.1`
5	`A00.9`	`A00.9`

RemoveNonTerminal

Funktionsparameter:

- `$data`
  Umsteiger-Einträge

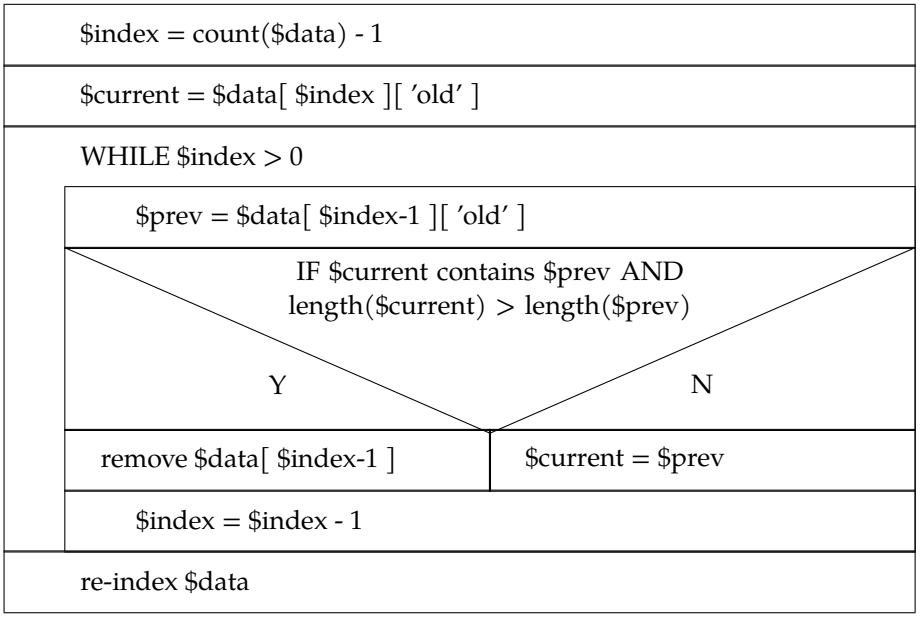
Lokale Variablen:

- `$index`
  Der Umsteiger-Eintrag, der aktuell verarbeitet wird.
  Die Funktion läuft vom letztem Eintrag zum ersten.
- `$current`
  Der alte Kode des aktuellen Umsteiger-Eintrags.
  Hierbei bedeutet $data[$index]['old'] Zugriff auf Zeile mit Index = Wert von $index und Spalte 'Alter Kode'. Zum Beispiel: data[4]['old'] = `A00.1`.
- `$prev`
  Der alte Kode des zuletzt verarbeiteten Umsteiger-Eintrags.
  Die Variable wird auf $current gesetzt, wenn der Eintrag *nicht* entfernt wird.

Anmerkungen:

- Die IF-Bedingung bezieht sich auf String-basierte Operationen; also contains: „enthält" als Sub-String und length: Anzahl der Zeichen im String.
- remove: Entfernt eine ganze Zeile.

[1] Der Pseudocode ist beschrieben in Struktogrammen nach (**nassishneid**). Die später vorkommenden Algorithmen zur Umsteiger-Suche enthalten viele Variablenzuweisungen, die abhängig von einer Bedingung sind und deren parallele Darstellung in Nassi-Shneidermann-Diagrammen übersichtlicher ist. Sie werden also im Sinne der Einheitlichkeit ebenfalls für den Algorithmus in diesem Abschnitt verwendet.



Beispiel-Array nach Durchlaufen des Algorithmus:

Index	old (Alter Kode)	new (Neuer Kode)
0	A00.0	A00.0
1	A00.1	A00.1
2	A00.9	A00.9

Nicht-endständige Kodes

Für die meisten Anwendungen sind eigentlich nur die endständigen Kodes relevant. Statt diese bei jeder Operation herauszufiltern, können beim einmaligen Einlesen der Daten auch einfach die nicht-endständigen Kodes ausgeschlossen werden. Zu diesem Zweck funktioniert der oben erklärte Algorithmus ebenfalls.

Integration zusätzlicher Informationen

Im nächsten Abschnitt wird erklärt, wie ermittelt wird, ob es zu einem Kode einer bestimmten Version Umsteiger in einer älteren oder neueren Version gibt. Um diese zusätzliche Information speichern zu können, werden die Kodes um eine Spalte erweitert.

3

Versionsübergreifende Umsteiger-Suche

Dieser Abschnitt behandelt die versionsübergreifende Suche nach Umsteigern. Es geht also zum einen darum zu bestimmen, ob ein bestimmter Kode einer bestimmten Version sich überhaupt verändert hat, sowie zum anderen darum, abhängig von einer Start- und Zielversion zu ermitteln, welche Überleitungen von Kodes möglich sind. Die hier aufgeführten Algorithmen sind unabhängig von einer konkreten Implementation verfasst.

§ Wichtig: 1 Umsteiger nur Veränderungen 2 Datenstrukturen

3.1 Allgemeine Funktionen

Grundlegend ist davon auszugehen, dass Funktionen zum Lesen der Daten vorliegen nach erfolgreicher Integration aus dem vorherigen Abschnitt.

Daten Lesen

Konkret könnten zum Beispiel die Inhalte der Kodes- und Umsteiger-Dateien mit folgenden Parametern ausgelesen werden.

readData

- `$system`
  Das Kodiersystem, beispielsweise als Konstanten definiert `'icd10gm'` und `'ops'`.
- `$version`
  Die Version, beziehungsweise Jahreszahl.
- `$code`
  Ein Kode, nach dem gesucht wird. In Bezug auf die Notation von ICD-10-GM und OPS ist es sinnvoll automatisch an rechter Stelle einen Wildcard zu implizieren, das heißt einen Platzhalter für beliebige, weitere Zeichen. Dann werden mit einem leeren Kode-Wert auch alle Daten für die angegebenen Parameter gelesen.

- `$type`
  Die Art an Daten, die gelesen werden sollen. Für die Suche der Umsteiger werden folgende benötigt – angegeben ebenfalls wieder als mögliche Konstanten:
  - `'kodes'`
    Kodes-Daten: Kode und Titel.
  - `'umsteiger'`
    Umsteiger-Daten: alter Kode und neuer Kode, sowie Information über die automatische Überleitbarkeit in beide Richtungen. Es werden die Umsteiger von der angegebenen auf die nächstältere Version abgefragt. Suche über $code = neuer Kode.
  - `'umsteiger_join'`
    Wie Umsteiger-Daten, aber zusätzlich mit Titel für jeweils den alten und neuen Kode. „Join“ bezieht sich auf die entsprechende Datenbankoperation.
  - `'umsteiger_join_alt'`
    Wie oben, außer dass die Suche über $code = alter Kode und in die andere Richtung erfolgt. Das heißt es wird die angegebene Version mit der nächstneueren Version verglichen. „Alt“ für „alternate“.

Nächste Version

Außerdem wird eine Funktion benötigt, die ausgehend von System und Version die jeweils nächstältere und -neuere Version ermittelt. Wenn diese nicht existiert, bleibt der Wert leer.

Hierfür kann die in §TODO: Verweis im Kommentar erwähnte strukturierte Datei dienen, welche die Versionen und Abweichungen definiert.

nextNewerVersion / nextOlderVersion

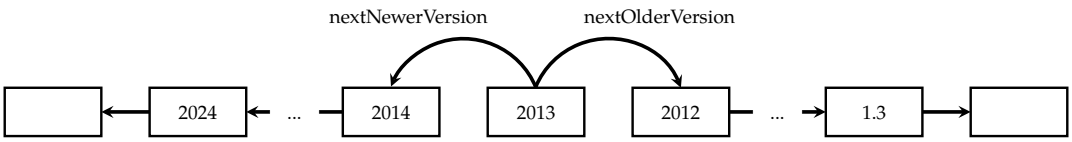


Abbildung 3.1: Ermitteln der nächstliegenden Versionen am Beispiel von ICD-10-GM.

Beispieldaten mit graphischer Darstellung

Um beispielsweise die Kodes M21.4 und M21.6 aus der ICD-10-GM Version 2016 abzufragen, sehen die Funktionsaufrufe dann so aus:

```
readData ('icd10gm', '2014', 'M21.4', 'kodes')
readData ('icd10gm', '2014', 'M21.6', 'kodes')
```

Und das Ergebnis als Tabelle:

code (Kode)	name (Titel)
M21.4	Plattfuß [Pes planus] (erworben)
M21.6	Sonstige erworbene Deformitäten des Knöchels und des Fußes

Für den restlichen Abschnitt wird angenommen, dass die Ergebnisse der readData-Funktion in einer zweidimensionalen Datenstruktur gespeichert werden die einer Tabelle ähneln, das heißt eine Array-Liste mit einem Eintrag pro Zeile beginnend mit Index = 0, sowie einem assoziativen Array (Map oder auch eine Klassenstruktur) mit Key = Spaltenüberschrift. Der Zugriff auf den Titel „Plattfuß“ in deren oberen Tabelle erfolgt dann über `$data[ 0 ][ 'name' ]`.

Analog zu den Kodes können die Umsteiger wie folgt abgefragt werden:

```
lies_daten ('icd10gm', '2014', 'M21.4', 'umsteiger')
lies_daten ('icd10gm', '2014', 'M21.6', 'umsteiger')
```

Das Ergebnis sieht wie folgt aus; “Auto” steht für die automatische Überleitbarkeit in die jeweilige Richtung, das heißt 2014←2013 und 2014→2013.

new (Kode Version 2014)	old (Kode Version 2013)	←Auto→	
M21.4	M21.4	A	A
M21.6	M21.6	A	A

Von Version 2014 auf 2013 ändern sich die angegebenen ICD-10-GM Kodes nicht. Wie bereits in der Datenintegration erwähnt, müssen diese Einträge gar nicht aufgenommen werden. Sie sind hier nur zur Vereinfachung aufgelistet, weil die Einträge auch so in den BfArM-Dateien enthalten sind. Bei der versionsübergreifenden Suche nach Umsteigern sind nur die Veränderungen relevant. Anstatt alle nicht relevanten Umsteiger-Einträge abzuspeichern und bei jeder Abfrage auszuschließen, kann alternativ auch angenommen werden, weil eine Suche nach Umsteigern immer von einem vorhanden Kode ausgeht, dass bei Nichtvorhandensein eines Umsteigers zwischen zwei Versionen dieser Kode einfach gleich bleibt.

Insgesamt treten bei M21.4 über alle Versionen keine Veränderungen in den Umsteigern auf. M21.6 hat allerdings folgende:

new (Kode Version 2015)	old (Kode Version 2014)	←Auto→	
M21.4	M21.4	A	A
M21.60	M21.6		A
M21.61	M21.6		A
M21.62	M21.6		A
M21.63	M21.6		A
M21.68	M21.6		A

new (Kode Version 2013)	old (Kode Version 2012)	←Auto→	
M21.4	M21.4	A	A
M21.6	M21.60	A	
M21.6	M21.67	A	
M21.6	M21.87	A	A

new (Kode Version 2.0)	old (Kode Version 1.3)	←Auto→	
M21.4	M21.4	A	A
M21.60	M21.6		A
M21.67	M21.6		A
M21.87	M21.8		A

Anmerkung: Von Kode M21.8, Version 1.3. ausgehend gibt es noch wesentlich mehr Umsteiger, aber diese werden hier nicht gelistet, weil sind von M21.6, Version 2014 aus nicht erreichbar sind.

Die Ergebnisse können als Graph dargestellt werden. Die Versionen sind horizontal angeordnet und die Kodes pro Version vertikal darunter als Knoten. Ausgangspunkt sind wie erwähnt die Kodes M21.4 und M21.6 der Version 2014. Die Knoten der anderen Versionen entsprechen den Kodes, die über Umsteiger erreichbar sind. Die Pfeilspitzen geben die Richtung der automatischen Überleitbarkeit an.

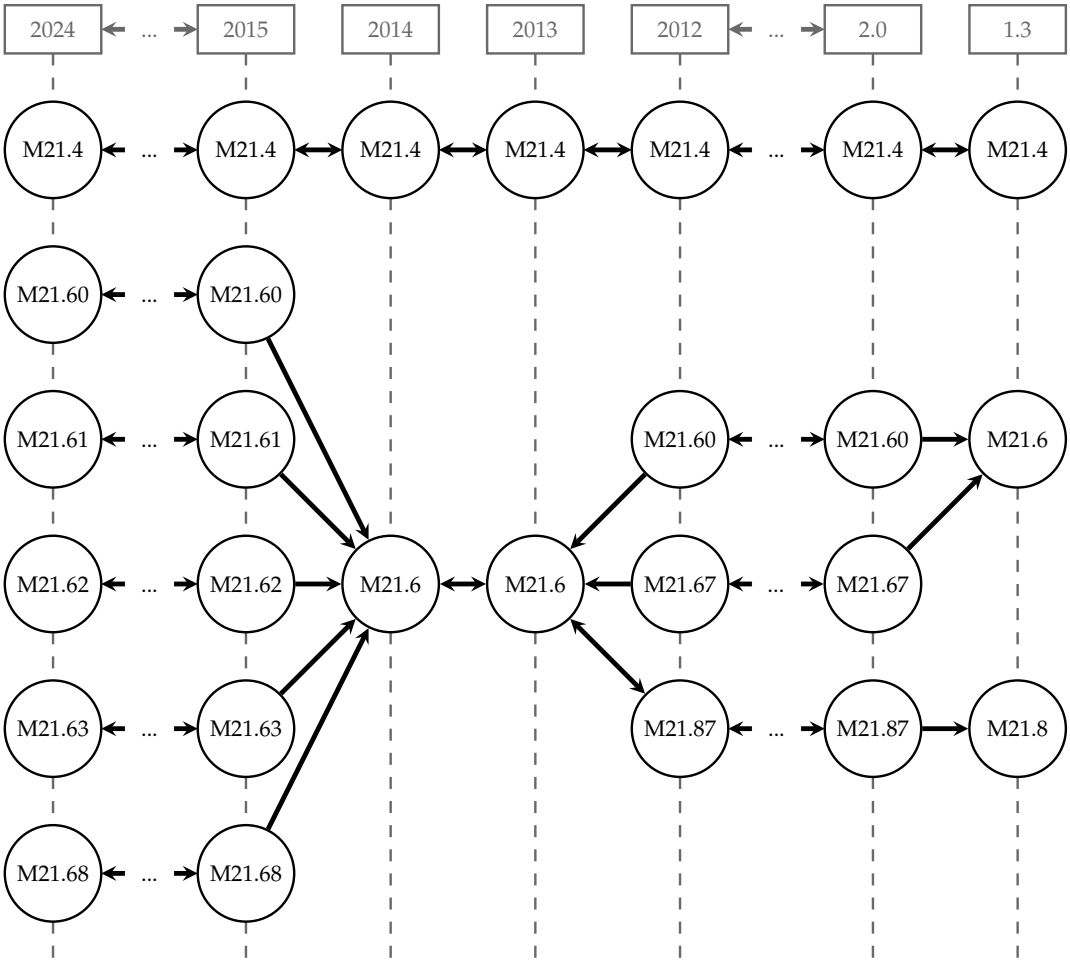


Abbildung 3.2: Graphische Repräsentation der Umsteiger ausgehend von den Kodes M21.4 und M21.6 der ICD-10-GM Version 2014.

3.2 Transitive Hülle

Wenn statt der graphischen Repräsentation oben:

1. Die nicht relevanten Umsteiger ausgeschlossen werden; also wie besprochen die Umsteiger-Einträge ohne Veränderung von Kodes.
2. Die Kanten im Graph die Suchrichtung anzeigen statt der automatischen Überleitbarkeit.

Dann ergibt sich ein gerichteter Graph. Das heißt ein Graph, der nur einfach gerichtete Kanten enthält und in dem jeder Knoten mit mindestens einem anderen Knoten über eine gerichtete Kante –auch Bogen genannt– verbunden ist. Die Suche nach über Umsteiger erreichbaren Kodes entspricht daher der Bestimmung der transitiven Hülle in der Graphentheorie.

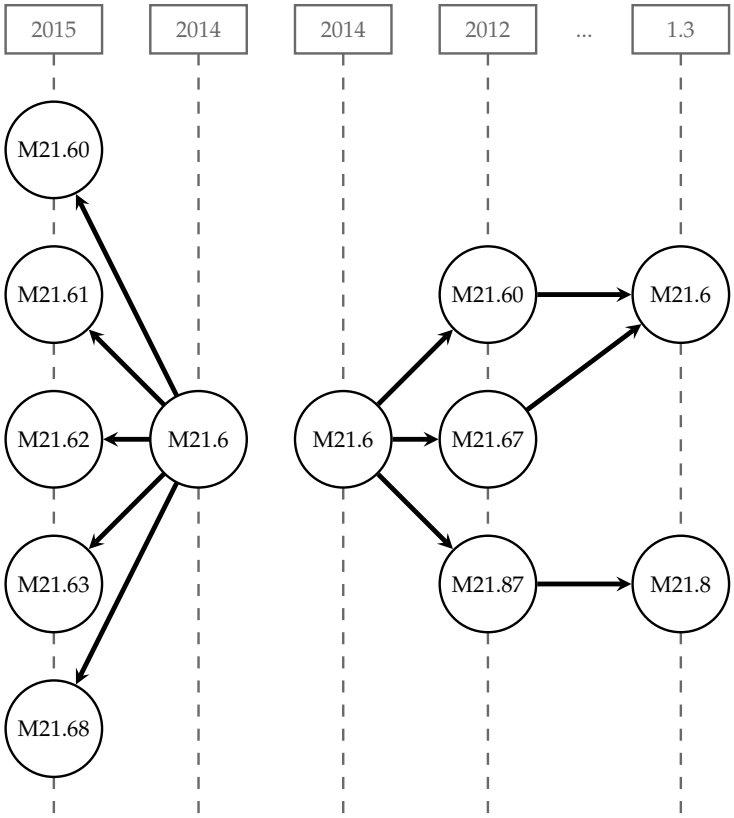


Abbildung 3.3: Von M21.6, Version 2014 erreichbare Kodes als gerichteter Graph.

Die transitive Hülle wird in (Gross, Yellen und Zhang, 2013, Seite 172) wie folgt definiert:

> **D26:** The ***transitive closure*** R^* of a binary relation R is the relation R^* defined by $(x, y) \in R^*$ if and only if there exists a sequence $x = v_0, v_1, v_2, \ldots, v_k = y$ such that $k \geq 1$ and $(v_i, v_{i+1}) \in R$, for $i = 0, 1, \ldots, k-1$. Equivalently, the transitive closure R^* of the relation R is the smallest transitive relation that contains R.
>
> **D27:** Let G be the digraph representing a relation R. Then the digraph G^* representing the transitive closure R^* of R is called the ***transitive closure of the digraph*** G. Thus, an arc (x, y), $x \neq y$, is in the transitive closure G^* if and only if there is a directed x-y path in G. Similarly, there is a self-loop in digraph D^* at vertex x if and only if there is a directed cycle in digraph G that contains x.

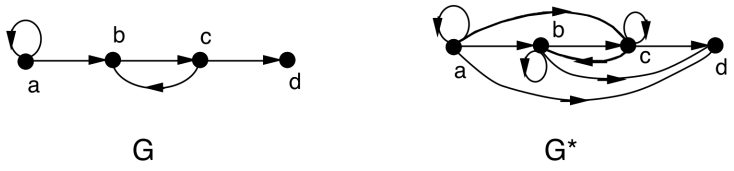
Oder kurz zusammengefasst: die transitive Hülle eines gerichteten Graphens G ist wiederum ein gerichteter Graph G^*, der von jedem Knoten einen Bogen zu allen von diesem Knoten aus in G erreichbaren Nachbarn besitzt.

In (Gross, Yellen und Zhang, 2013, Seite 172) ist hierzu folgendes Beispiel dargestellt:

E7: Suppose a relation R on the set $S = \{a, b, c, d\}$ is given by

$$\{(a, a), (a, b), (b, c), (c, b), (c, d)\}$$

Then the digraph G representing the relation R and the transitive closure G^* are as shown in Figure 3.1.7.



Es gibt mehrere Verfahren, um die transitive Hülle zu bestimmen. Es werden nun zwei Algorithmen vorgestellt, die anhand ihrer primären Suchrichtung benannt sind. Wie in den Graphen dargestellt, läuft die *horizontale* Suche primär über die Versionen –mit jeweils einzelnen Kodes– und die *vertikale* Suche primär über die Kodes –es werden zuerst alle Kodes einer Version gelesen.

Horizontale Suche

Ein intuitives Verfahren die transitive Hülle zu bestimmen wird in (Jakobsson, 1991, Seite 200) so beschrieben:

2.2 Search

A second approach for computing the transitive closure is to search the graph n times, each time starting from a different node, thereby determining what can be reached from that node. In this approach, we completely disregard any parts of the TC-relation that have already been computed for other starting nodes. Instead, we search the entire part of the graph that is reachable from the starting node by following every edge we can get to.

Ähnlich wie vorherigen Abschnitt für den Kode M24.6 dargestellt, werden also einfach ausgehend von einem Knoten so viele Suchen gestartet bis keine benachbarten Knoten mehr gefunden werden – oder im Anwendungsfall der Umsteiger bis alle Versionen verarbeitet wurden.

Ergebnisse für andere Knoten, beziehungsweise Kodes, werden ignoriert. Das hieße im Beispiel oben, dass eine rückwärts chronologische Suche nach Umsteigern von M21.60 und M21.61 der ICD-10-GM Version ab dem Vorgänger M21.6 der Version 2014 zweimal exakt gleich ablaufen würde.

Der Pseudocode für diese Vorgehensweise:

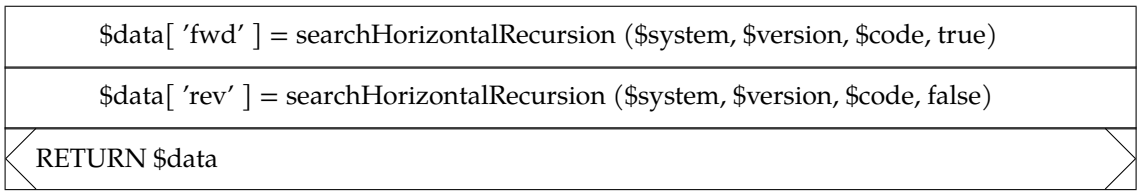
searchHorizontal

Funktionsparameter:

– `$system`, `$version`, `$code`
Wie in Funktion 3.1 readData.

Lokale Variable:

– `$data` Rückgabewert, initial: leer.
Eine assoziative Datenstruktur mit Key `fwd` ⇒ Umsteiger, die chronologisch vorwärts von dem Suchkode erreichbar sind und `rev` ⇒ chronologisch rückwärts.



Ausgehend von einem gegebenen Kode einer Version eines Kodiersystems wird chronologisch vorwärts und rückwärts eine rekursive Suche über alle Versionen gestartet.

searchHorizontalRecursion

Funktionsparameter:

– `$system`, `$version`, `$code`
Wie searchHorizontal.

– `$chronological`
Bestimmt die Suchrichtung. TRUE: chronologisch vorwärts und FALSE: rückwärts.

Lokale Variablen:

– `$data` Rückgabewert, initial: leer.
Eine mehrdimensionale, assoziative Datenstruktur, die die Ergebnisse der rekursiven Suche enthält. Gefundene Umsteiger werden als Liste mit dem Key `umsteiger` gespeichert; zusätzlich zu der Listen auch die Versionen vorher und nachher. Jeder einzelne Umsteiger-Eintrag enthält wiederum die Informationen zur automatischen Überleitbarkeit, sowie die Titel der Kodes vorher und nachher. Falls ein Umsteiger weitere Umsteiger hat, werden diese unter dem einem Key `recursion` abgelegt und die Suche fortgesetzt. Das Beispielergebnis für M21.4 befindet sich im Anhang A.2.

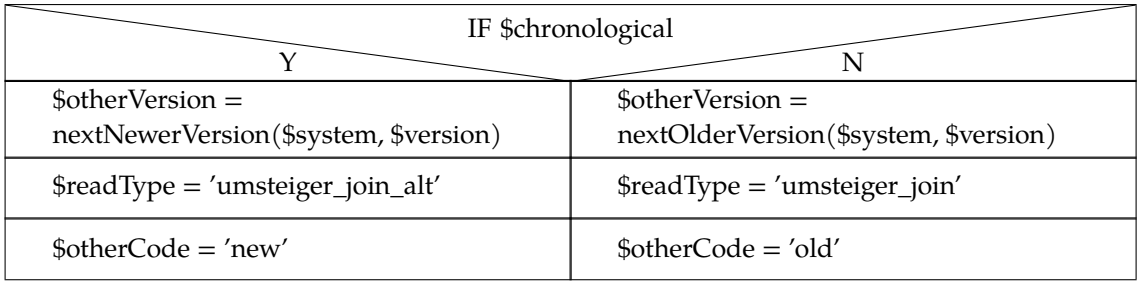
– `$otherVersion`
Die Version, zu der die Umsteiger ermittelt werden.

– `$otherCode`
Die Kode, zu dem der aktuelle Kode übergeleitet wird.

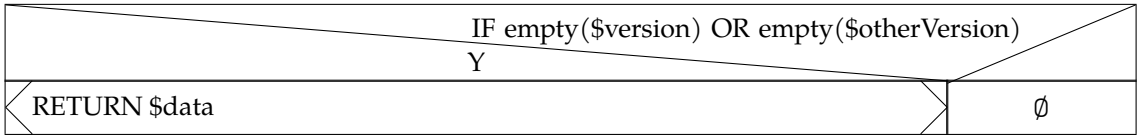
– `$readType`
Auf welche Art die Daten gelesen werden; siehe `readData`.

– `$umsteiger`, `$entry`
Liste der Umsteiger, Schleifenvariable: Umsteiger-Eintrag.

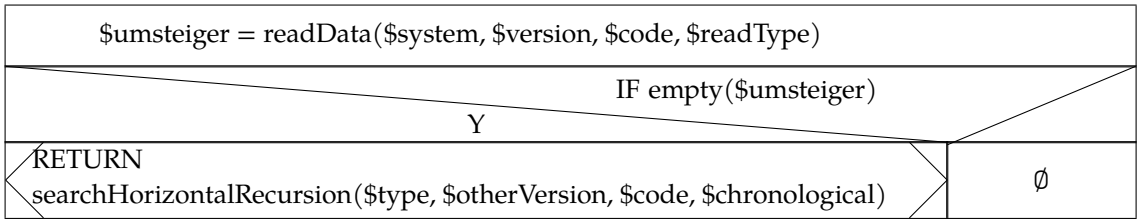
– `$recursion`
  Ergebnis des rekursiven Funktionsaufrufs bei gefundenen Umsteigern.



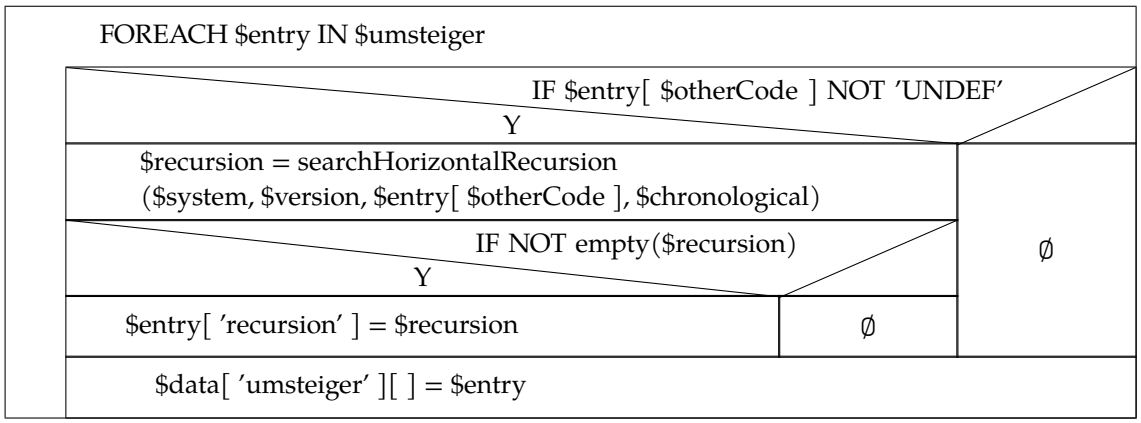
Lokale Variablen werden anhand der Suchrichtung unterschiedlich belegt.



Falls es in der Suchrichtung keine weitere Version mehr gibt, endet die Rekursion.



Die Umsteiger zwischen den Versionen im aktuellen Rekursionsschritt werden ermittelt. Falls es keine gibt –also keine, die eine Veränderung ausdrücken– dann wird die Rekursion mit der nächsten Version fortgesetzt.



Für jeden Umsteiger wird die Rekursion mit dem veränderten Kode fortgesetzt. Allerdings nur falls dieser nicht UNDEF ist, weil es sich dann um einen entfernten oder neu hinzugefügten Kode handelt. Also wenn wie im Beispiel der M21.6 der ICD-10-GM Version 2013 die Umsteiger [M21.60, M21.67, M21.87] in der Version 2012 hat, dann wird die Suche in die rückwärts chronologische Richtung mit diesen drei Kodes fortgesetzt statt mit M21.6. Die Ergebnisse dieser Verzweigung, sofern vorhanden, werden abgespeichert mit dem Key `recursion` zusätzlich zu je-

dem Umsteiger-Eintrag. Die Liste aller Umsteiger-Einträge, inklusive der UNDEF-Umsteiger, wird in das Ergebnis mit dem Key `umsteiger` aufgenommen.

```
$data[ 'version' ] = $version
$data[ 'other' ] = $otherVersion
RETURN $data
```

Falls es Umsteiger für einen Kode gibt, werden zusätzlich die Versionen vorher/-nachher gespeichert. Dann endet auch hier die Rekursion.

Appendix A.2 enthält das Ergebnis im JSON-Format für den Beispielaufruf der horizontale Suche für M21.6, Version 2014 entsprechend der Abbildung 3.3.

Vertikale Suche

Wie bereits erwähnt startet der Algorithmus der horizontalen Suche für jeden Kode bei Null. Gerade wenn die Umsteiger für alle Kodes einer Version – beziehungsweise eines Kodiersystems– gefunden werden sollen, ist diese Vorgehensweise nicht effizient.

Besonders für gerichtete Graphen ist allerdings Purdoms Algorithmus gut geeignet, um die transitive Hülle zu bestimmen. Folgende kurze Beschreibung des Algorithmus basiert auf (Purdom, 1970) und (Dar, 1993, Seite 77):

1. Ausgehend von einem Graphen G: Bestimme die stark zusammenhängenden Komponenten (SCC) in G und vereinige diese in jeweils einen einzelnen Knoten. Das ergibt einen zyklenfreien, verdichteten Graphen G_c.
2. Sortiere G_c topologisch.
3. Ermittele die transitive Hülle von G_c über dessen Adjazenzlisten in rückwärts topologischer Reihenfolge.
4. Bestimme die transitive Hülle des ursprünglichen Graphens G über die Nachbarschaftslisten der stark zusammenhängenen Komponenten in G_c: ein Knoten y ist ein erreichbarer Nachbar von Knoten x, falls $SCC(y) = SCC(x)$ oder falls $SCC(y)$ ein erreichbarer Nachbar von $SCC(x)$ ist.

Die Vereinigung der stark zusammenhängenden Komponenten aus Schritt eins wird in (Purdom, 1970) wie folgt dargestellt:

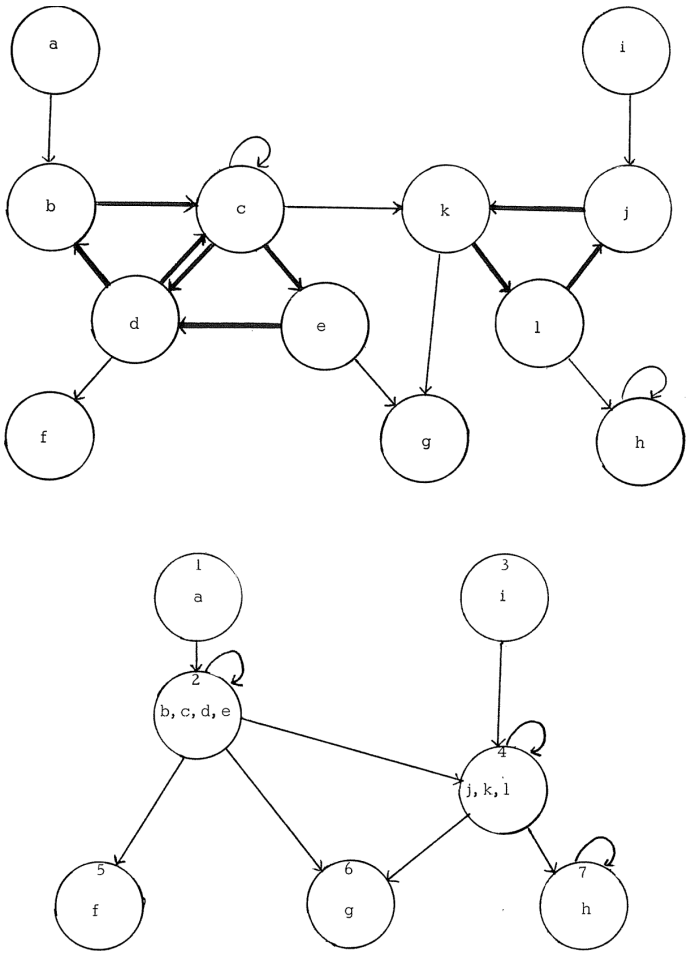


Abbildung 3.4: Beispielgraph G und Vereinigung der stark zusammenhängenden Komponenten aus (Purdom, 1970, Seite 78).

Die hier vorgestellte vertikale Suche nach Umsteigern entspricht nicht ganz Purdoms Algorithmus.

Zum Beispiel ist die topologische Sortierung aus Schritt zwei in der Anwendung der Suche nach Umsteigern nicht notwendig, weil die Versionen immer in einer bestimmte Richtung verglichen werden und die Kodes sich nur zwischen zwei Versionen ändern können. Die Daten sind also schon implizit topologisch sortiert.

Aber die vertikale Suche verwendet wesentlich zwei Besonderheiten von Purdoms Algorithmus, die in (Dar, 1993, Seite 76f) hervorgehoben werden:

5.3.1 The Purdom Algorithm

Purdom, in [Pur70], made two key observations:

1. During the computation of transitive closure of a directed acyclic graph, if node $s \prec_t t$, then additions to the successor list of node s cannot affect the successor list of node t. One should therefore compute the successor list of t first and then that of s. By processing nodes in reverse topological order, one need add to a node only the successor lists of its immediate successors since the latter would already have been fully expanded. We call this idea the *immediate successor* optimization [AgJ90].

2. All nodes within a strongly connected component (SCC) in a graph have identical reachability properties, and the condensation graph obtained by collapsing all the nodes in each strongly connected component into a single node is acyclic.

Konkret bedeuten diese Besonderheiten für die Umsteiger-Suche:

1. Topologisch rückwärts gerichtete Vorgehensweise: So ist zum Beispiel bei der Suche nach den Umsteigern eines ICD-GM-10 Kode der Version 1.3 die Reihenfolge, in der die anderen Versionen abgearbeitet werden: 2024, 2023, 2022 und so weiter. Oder anders ausgedrückt: Wenn die Überleitungen in chronologischer Reihenfolge bestimmt werden sollen, dann läuft die vertikale Suche umgekehrt, also chronologisch rückwärts.

2. Vereinigung der stark zusammenhängenden Komponenten: Nach jedem Schritt über eine Version, wird geprüft, ob ein neu gefundener Umsteiger eine Überleitung in einen Kode enthält, der bereits in den davor abgearbeitet Versionen als Umsteiger vorkommt. Falls ja, dann werden diese Umsteiger vereinigt, das heißt die Zwischenschritte werden zu einem Schritt zusammengefügt.

Einfaches Beispiel für die Vereinigung:

Angenommen ein abstraktes System hat Umsteiger von der Version 1.0 auf 2.0: $A \rightarrow B$ und $M \rightarrow N$, sowie von Version 2.0 auf 3.0: $B \rightarrow C$. Wenn nun in chronologischer Reihenfolge 1.0→2.0→3.0 Umsteiger ermittelt werden sollen, erfolgt die Suche sowie Vereinigung gefundener Umsteiger in umgekehrter Reihenfolge. Das heißt wenn im Schritt 3.0←2.0 der Umsteiger $C \leftarrow B$ gefunden wird, dann wird danach bei der Vereinigung mit Version 1.0 der Umsteiger $B \leftarrow A$ in $C \leftarrow A$ umgewandelt. A in Version 1.0 entspricht C in Version 3.0.

Falls es Verzweigungen gibt, also es mehrere Umsteiger für einen Kode gibt, dann werden entsprechend Listen vereinigt. Ein reales Beispiel dazu folgt später.

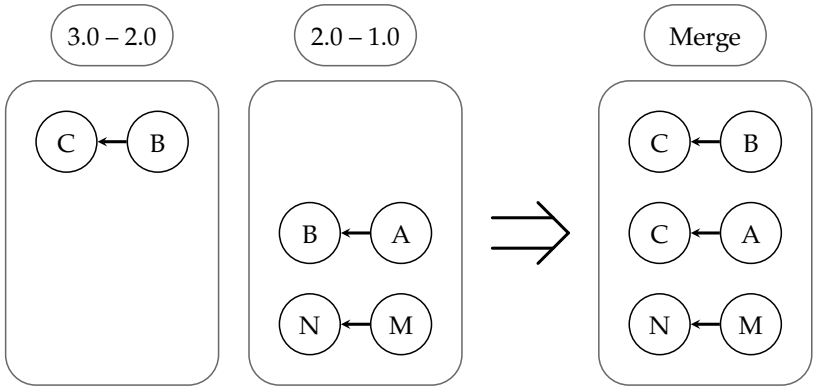


Abbildung 3.5: Graphische Veranschaulichung der zwei Besonderheiten von Purdoms Algorithmus, die in der vertikalen Suche zur Anwendung kommen.

Die gesamte Algorithmus für die vertikale Suche als Pseudocode:

searchVertical

Funktionsparameter:

- `$system`
  Das Kodiersystem, beispielsweise als Konstanten definiert `'icd10gm'` und `'ops'`.
- `$targetVersion`
  Die Zielversion, auf die von allen anderen Version die Umsteiger über alle Kodes ermittelt werden.
- `$function`
  Eine optionale, anonyme Funktion.

Lokale Variable:

- `$data` Rückgabewert, initial: leer.
  Eine zweidimensionale, assoziative Datenstruktur – wird nur befüllt wenn `$function` nicht gesetzt ist. Die erste Dimension hat als Keys alle Versionen außer der Zielversion. Die zweite Dimension hat als Keys die Kodes der Version und als Value die Liste der zugehörigen Umsteiger, inklusive Vereinigung. Kodes ohne Umsteiger werden nicht aufgenommen.

```
$data += searchVerticalSubroutine($system, $targetVersion, false, $function)
$data += searchVerticalSubroutine($system, $targetVersion, true, $function)
RETURN $data
```

Basierend auf der Zielversionen wird in beide Richtungen die vertikale Suche als Unterfunktion gestartet.

`$data +=` bedeutet, dass die assoziativen Datenstrukturen zusammengefasst werden. Wenn ein Key auf beiden Seiten der Addition existiert, wird der von der linken Seite für die Summe übernommen. Das kann allerdings hier bei den Versionen als Keys nicht passieren, weil jede Version nur einmal bearbeitet wird.

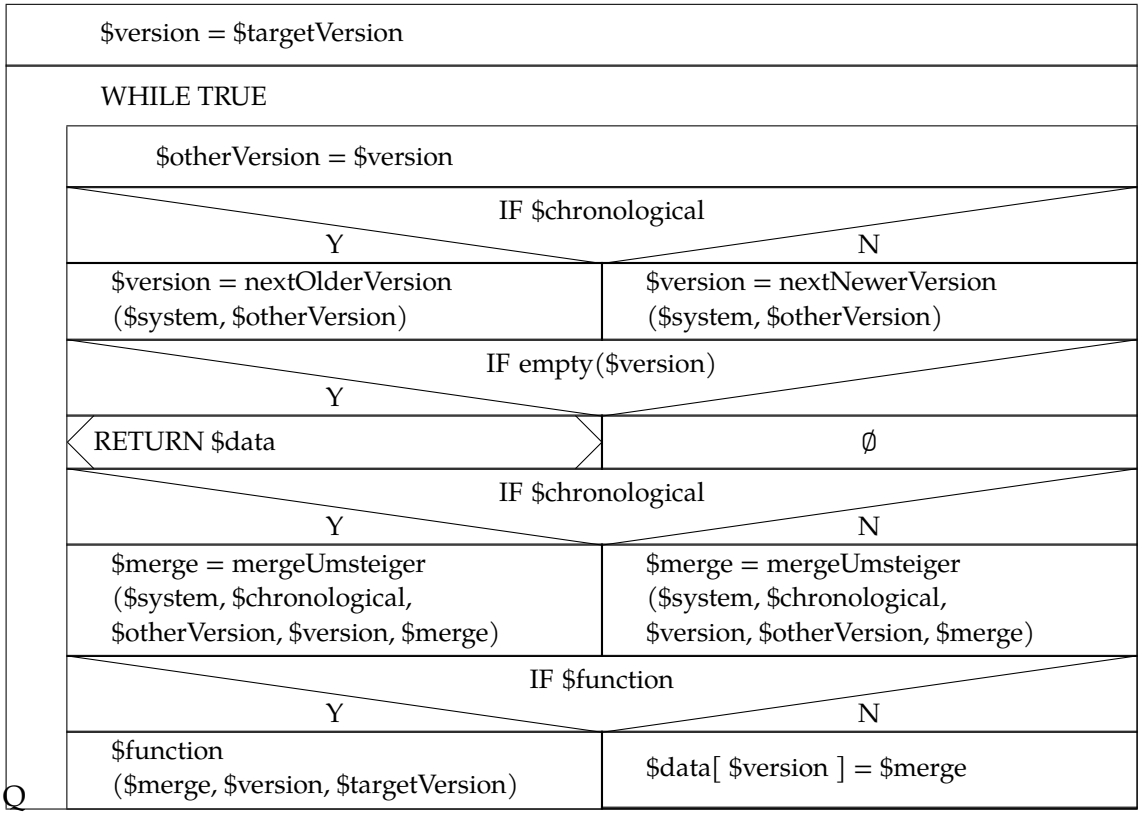
searchVerticalSub

Funktionsparameter:

- `$system`, `$targetVersion`, `$function`
  Wie searchVertical.
- `$chronological`
  Bestimmt die Richtung, in der die Versionen abgearbeitet werden. TRUE: chronologisch vorwärts und FALSE: rückwärts. Die Reihenfolge ist wie erwähnt topologisch rückwärts, das heißt ausgehend von der Zielversion.

Lokale Variablen:

- `$data` Rückgabewert, initial: leer.
  Eine zweidimensionale, assoziative Datenstruktur – wird nur befüllt wenn `$function` nicht gesetzt ist! Die erste Dimension hat als Keys alle Versionen außer der Zielversion. Die zweite Dimension hat als Keys die Kodes aller bisher verarbeiteten Umsteiger und als Values die Listen der Kodes, in die übergeleitet wird, inklusive Vereinigungen wie oben beschrieben. Das heißt im Gegensatz zur horizontalen Suche werden die Umsteiger als Key-Value Paare gespeichert: Kode der aktuellen Version ⇒ Kode/s in die Vergleichsversion. Kodes ohne Umsteiger werden nicht aufgenommen.
- `$merge`
  bla
- `$version`
  bla
- `$otherVersion`
  bla



mergeUmsteiger

Funktionsparameter:

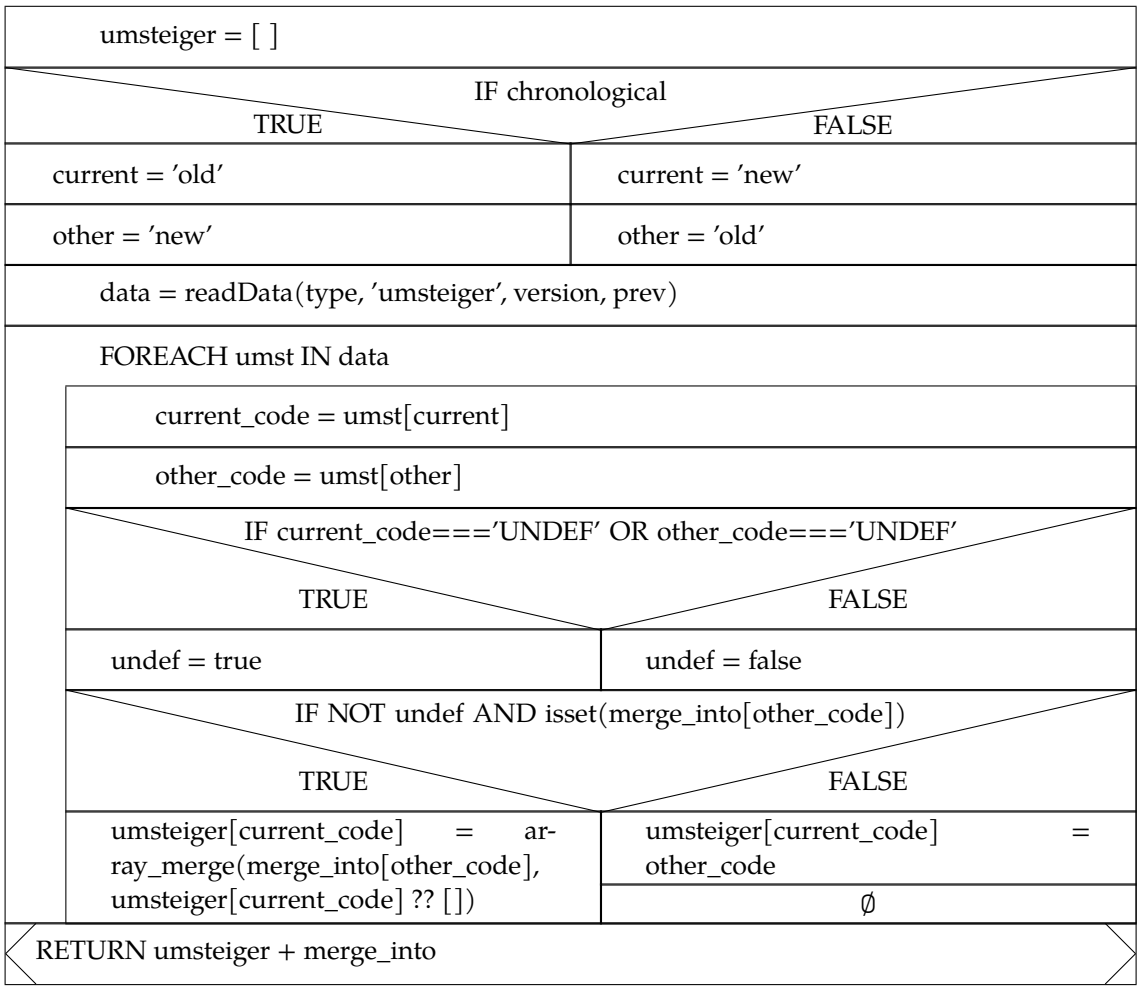
- `$system`, `$target`, `$function`
  Wie searchVertical.
- `$chronological`
  Bestimmt die Richtung, in der die Versionen abgearbeitet werden. TRUE: chronologisch vorwärts und FALSE: rückwärts. Die Reihenfolge ist wie erwähnt topologisch rückwärts, das heißt ausgehend von der Zielversion.

§TODO: ab hier weitermachen

Lokale Variable:

- `$data` Rückgabewert, initial: leer.
  Eine zweidimensionale, assoziative Datenstruktur – wird nur befüllt wenn `$function` nicht gesetzt ist. Die erste Dimension hat als Keys alle Versionen außer der Zielversion. Die zweite Dimension hat als Keys die Kodes der Version und als Value die Liste der zugehörigen Umsteiger, inklusive Vereinigung. Kodes ohne Umsteiger werden nicht aufgenommen.
- `$merge`
  bla, initial: leer.

– `$version`
  bla, initial = target.



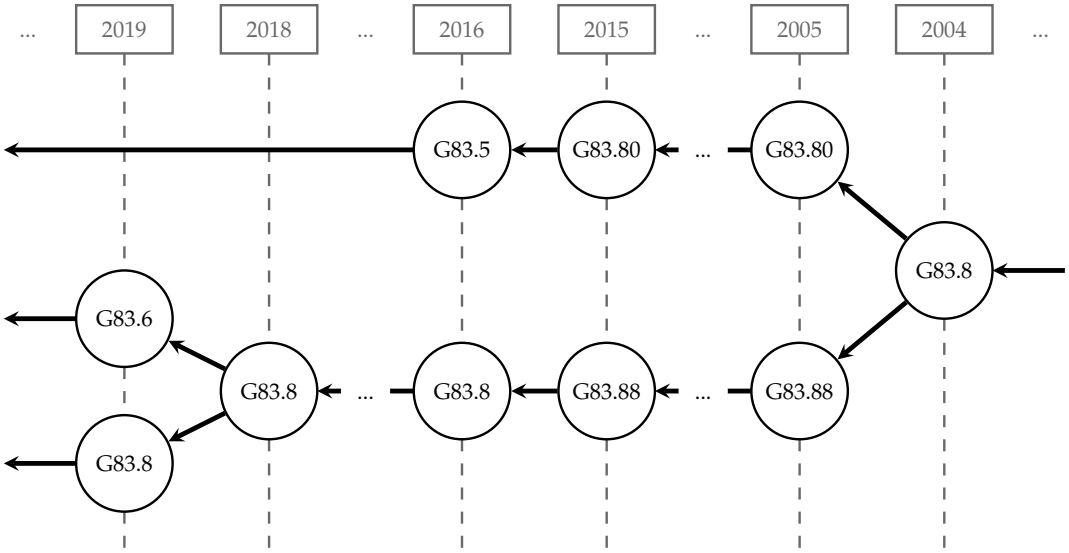


Abbildung 3.6: caption

test1 test2

3.3 Vergleich der Suchalgorithmen

Wenn die Daten aus einer Datenbank gelesen werden, dann ist der Menge pro readData Aufruf weniger entscheidend als die Anzahl solcher Aufrufe. Das heißt alle Umsteiger einer Version auf einmal zu lesen ist wesentlich schneller als sie einzeln zu lesen.

Außerdem redundant

In dem Fall ist der

Horizontal:

Schneller bei einem Kode

Mehr Daten hinzufügbar. Vertikale Suche mit Titeln würde schnell mehrere hundert MB groß werden, Arbeitsspeicher.

Vertikal:

Schneller bei allen Kodes

Verwendung für ConceptMap

Verwendung für Bestimmen ob ein Kodes Umsteiger haben für alle Kodes und Versionen

3.4 Schreiben der ConceptMap

Appendix XML A.3

4

Web-Applikation

5

Zusammenfassung

5.1 Ergebnisse

5.2 Diskussion

5.3 Ausblick

Literatur

Braunstein, M. (2022). Health Informatics on FHIR: How HL7's API is Transforming Healthcare. Health Informatics. Springer International Publishing. ISBN: 9783030915636.

Dar, S. (1993). Augmenting Databases with Generalized Transitive Closure. Computer Sciences Technical Report. University of Wisconsin–Madison.

Fielding, R.T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral Dissertation. University of California, Irvine. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm (besucht am 04. 10. 2024).

Gross, J., Yellen, J. und Zhang, P. (2013). Handbook of Graph Theory, Second Edition. Discrete Mathematics and Its Applications. Taylor & Francis. ISBN: 9781439880180.

Heckmann, S. (2022). Digitalstrategie im Krankenhaus: Einführung und Umsetzung von Datenkompetenz und Compliance. Springer Fachmedien Wiesbaden. Kap. HL7® FHIR® als Grundlage für moderne Digitalstrategien, S. 307–331. ISBN: 9783658362263. DOI: 10.1007/978-3-658-36226-3_21.

ISO 12300 (2014). *Health Informatics – Principles of Mapping between Terminological Systems*. International Organization for Standardization. URL: https://www.iso.org/standard/51344.html (besucht am 21. 10. 2024).

ISO 21564 (2019). *Health Informatics – Terminology Resource Map Quality Measures (MapQual)*. International Organization for Standardization. URL: https://www.iso.org/standard/71088.html (besucht am 21. 10. 2024).

Jakobsson, H. (1991). Mixed-Approach Algorithms for Transitive Closure. In Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, S. 199–205. URL: https://dl.acm.org/doi/pdf/10.1145/113413.113431.

Philipp, P., Oliveira, J. Veloso de, Appenzeller, A., Hartz, T. und Beyerer, J. (Dez. 2022). Evaluation of an Automated Mapping from ICD-10 to SNOMED CT. In 2022 International Conference on Computational Science and Computational Intelligence, S. 1604–1609. DOI: 10.1109/CSCI58124.2022.00287.

Purdom, P. (1970). A Transitive Closure Algorithm. BIT Numerical Mathematics *10*, 76–94.

Saripalle, R. (2019). Representing UMLS knowledge using FHIR terminological resources. In 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), IEEE, S. 1109–1112. DOI: 10.1109/BIBM47256.2019.8983305.

A

Appendix

A.1 Abweichungen zwischen ICD-10-GM und OPS Versionen

ICD-10-GM

Version	Abweichungen zwischen den Versionen	
2025	URL	`version2025-vorab/icd10gm2025syst-ueberl-vorab.zip`
	Kodes	`Klassifikationsdateien/icd10gm2025syst_vorab.txt`
	Umsteiger	`Klassifikationsdateien/icd10gm2025syst_umsteiger_2024_2025_vorab.txt`
	Sonstiges	· Vorab-Version
2024	URL	`version2024/icd10gm2024syst-ueberl.zip`
	Umsteiger	`Klassifikationsdateien/icd10gm2024syst_umsteiger_2023_20221206_2024.txt`
2023	URL	`version2023/icd10gm2023syst-ueberl_20221206.zip`
	Kodes	`Klassifikationsdateien/icd10gm2023syst_20221206.txt`
	Umsteiger	`Klassifikationsdateien/icd10gm2023syst_umsteiger_2022_2023_20221206.txt`
2022	URL	`vorgaenger/icd10gm2022.zip`
	Sonstiges	· Zip-Unterdatei: `icd10gm2022syst-ueberl.zip`
2021	URL	`vorgaenger/icd10gm2021.zip`
	Verzeichnis	`icd10gm2021syst-ueberl-20201111`
2020	URL	`vorgaenger/icd10gm2020.zip`
	Verzeichnis	`icd10gm2020syst-ueberl`
2019	URL	`vorgaenger/icd10gm2019.zip`
	Verzeichnis	`icd10gm2019syst-ueberl`

2018	URL	`vorgaenger/icd10gm2018.zip`
	Verzeichnis	`x1gut2018`
2017	URL	`vorgaenger/icd10gm2017.zip`
	Verzeichnis	`x1gut2017`
2016	URL	`vorgaenger/icd10gm2016.zip`
	Verzeichnis	`x1gut2016`
2015	URL	`vorgaenger/icd10gm2015.zip`
	Verzeichnis	`x1gut2015`
2014	URL	`vorgaenger/icd10gm2014.zip`
	Verzeichnis	`x1gua2014`
2013	URL	`vorgaenger/icd10gm2013.zip`
	Verzeichnis	`x1gua2013`
2012	URL	`vorgaenger/icd10gm2012.zip`
	Kodes	`x1ueb2011_2012/Klassifikationsdateien/icd10gmsyst2012.txt`
	Umsteiger	`x1ueb2011_2012/Klassifikationsdateien/umsteiger_icd10gmsyst2011_icd10gmsyst2012.txt`
2011	URL	`vorgaenger/icd10gm2011.zip`
	Kodes	`x1ueb2010_2011/Klassifikationsdateien/icd10gmsyst2011.txt`
	Umsteiger	`x1ueb2010_2011/Klassifikationsdateien/umsteiger_icd10gmsyst2010_icd10gmsyst2011.txt`
2010	URL	`vorgaenger/icd10gm2010.zip`
	Kodes	`x1ueb2009_2010/Klassifikationsdateien/icd10gmsyst2010.txt`
	Umsteiger	`x1ueb2009_2010/Klassifikationsdateien/umsteiger_icd10gmsyst2009_icd10gmsyst2010.txt`
2009	URL	`vorgaenger/icd10gm2009.zip`
	Kodes	`x1ueb2008_2009/Klassifikationsdateien/icd10gmsyst2009.txt`
	Umsteiger	`x1ueb2008_2009/Klassifikationsdateien/umsteiger_icd10gmsyst2008_icd10gmsyst2009.txt`
2008	URL	`vorgaenger/icd10gm2008.zip`
	Kodes	`x1ueb2007_2008/Klassifikationsdateien/icd10v2008.txt`
	Umsteiger	`x1ueb2007_2008/Klassifikationsdateien/umsteiger20072008.txt`
	Sonstiges	· ISO-8859-1

2007	URL	`vorgaenger/icd10gm2007.zip`
	Kodes	`x1ueb2006_2007/Klassifikationsdateien/ICD10V2007.txt`
	Umsteiger	`x1ueb2006_2007/Klassifikationsdateien/Umsteiger.txt`
	Sonstiges	· ISO-8859-1
2006	URL	`vorgaenger/icd10gm2006.zip`
	Kodes	`x1ueb2005_2006/ICD10V2006.txt`
	Umsteiger	`x1ueb2005_2006/umsteiger.txt`
	Sonstiges	· ISO-8859-1
2005	URL	`vorgaenger/icd10gm2005.zip`
	Kodes	`x1ueb2004_2005/ICD10V2005.txt`
	Umsteiger	`x1ueb2004_2005/umsteiger.txt`
	Sonstiges	· ISO-8859-1
2004	URL	`vorgaenger/icd10gm2004.zip`
	Kodes	`x1ueb20_2004/icd10v2004.txt`
	Umsteiger	`x1ueb20_2004/Umsteiger.txt`
	Sonstiges	· ISO-8859-1 · Punkt-Strich-Notation · 6-Spalten-Umsteiger
2.0	URL	`vorgaenger/icd10gm20.zip`
	Kodes	`x1ueb13_20_v11/icd10v20.txt`
	Umsteiger	`x1ueb13_20_v11/Umsteiger.txt`
	Sonstiges	· ISO-8859-1 · Punkt-Strich-Notation · Kreuz-Stern-System · 6-Spalten-Umsteiger · Nicht endständige Umsteiger
1.3	Kodes	`x1ueb13_20_v11/icd10v13.txt`
	Sonstiges	· ISO-8859-1 · Punkt-Strich-Notation · Kreuz-Stern-System · Keine Überleitung

OPS

Version	Abweichungen zwischen den Versionen	
2025	URL	`version2025-vorab/ops2025syst-ueberl-vorab.zip`
	Kodes	`Klassifikationsdateien/ops2025syst_vorab.txt`
	Umsteiger	`Klassifikationsdateien/ops2025syst_umsteiger_2024_2025_vorab.txt`
	Sonstiges	· Vorab-Version
2024	URL	`version2024/ops2024syst-ueberl.zip`
2023	URL	`version2023/ops2023syst-ueberl.zip`
2022	URL	`vorgaenger/ops2022.zip`
	Sonstiges	· Zip-Unterdatei: `ops2022syst-ueberl.zip`
2021	URL	`vorgaenger/ops2021.zip`
	Verzeichnis	`ops2021syst-ueberl`
2020	URL	`vorgaenger/ops2020.zip`
	Verzeichnis	`ops2020syst-ueberl`
2019	URL	`vorgaenger/ops2019.zip`
	Verzeichnis	`ops2019syst-ueberl`
2018	URL	`vorgaenger/ops2018.zip`
	Verzeichnis	`p1sut2018`
2017	URL	`vorgaenger/ops2017.zip`
	Verzeichnis	`p1sut2017`
2016	URL	`vorgaenger/ops2016.zip`
	Verzeichnis	`p1sut2016`
2015	URL	`vorgaenger/ops2015.zip`
	Verzeichnis	`p1sut2015`
2014	URL	`vorgaenger/ops2014.zip`
	Kodes	`p1sua2014-20131104/Klassifikationsdateien/ops2014syst_20131104.txt`
	Umsteiger	`p1sua2014-20131104/Klassifikationsdateien/ops2014syst_umsteiger_2013_2014_20131104.txt`
2013	URL	`vorgaenger/ops2013.zip`
	Kodes	`p1sua2013/Klassifikationsdateien/ops2013syst_20121113.txt`
	Umsteiger	`p1sua2013/Klassifikationsdateien/ops2013syst_umsteiger_2012_2013_20121109.txt`

2012	URL	`vorgaenger/ops2012.zip`
	Kodes	`p1ueb2011_2012/Klassifikationsdateien/opssyst2012.txt`
	Umsteiger	`p1ueb2011_2012/Klassifikationsdateien/umsteiger_opssyst2011_opssyst2012.txt`
2011	URL	`vorgaenger/ops2011.zip`
	Kodes	`p1ueb2010_2011/Klassifikationsdateien/opssyst2011.txt`
	Umsteiger	`p1ueb2010_2011/Klassifikationsdateien/umsteiger_opssyst2010_opssyst2011.txt`
2010	URL	`vorgaenger/ops2010.zip`
	Kodes	`p1ueb2009_2010/Klassifikationsdateien/opssyst2010.txt`
	Umsteiger	`p1ueb2009_2010/Klassifikationsdateien/umsteiger_opssyst2009_opssyst2010.txt`
2009	URL	`vorgaenger/ops2009.zip`
	Kodes	`p1ueb2008_2009/Klassifikationsdateien/opsamtl2009.txt`
	Umsteiger	`p1ueb2008_2009/Klassifikationsdateien/umsteigeramtl20082009.txt`
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)
2008	URL	`vorgaenger/ops2008.zip`
	Kodes	`ops2008amtl/p1ueb2007_2008/Klassifikationsdateien/opsamtl2008.txt`
	Umsteiger	`ops2008amtl/p1ueb2007_2008/Klassifikationsdateien/umsteigeramtl20072008.txt`
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)
2007	URL	`vorgaenger/ops2007.zip`
	Kodes	`ops2007amtl/p1ueb2006_2007/Klassifikationsdateien/opsamtl2007.txt`
	Umsteiger	`ops2007amtl/p1ueb2006_2007/Klassifikationsdateien/UmsteigerAmtlich.txt`
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)

2006	URL	`vorgaenger/ops2006.zip`
	Kodes	`ops2006amtl/p1ueb2005_2006/opsv2006.txt`
	Umsteiger	`ops2006amtl/p1ueb2005_2006/umsteiger.txt`
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 6-Spalten-Umsteiger (altes Format)
2005	URL	`vorgaenger/ops2005.zip`
	Kodes	`ops2005amtl/p1ueb2004_2005_v10/OPS2005.txt`
	Umsteiger	`ops2005amtl/p1ueb2004_2005_v10/umsteiger.txt`
	Sonstiges	· ISO-8859-1 · None statt UNDEF · 5-Spalten-Umsteiger
2004	URL	`vorgaenger/ops2004.zip`
	Kodes	`ops2004amtl/p1ueb21_2004_v10/opsv2004.txt`
	Umsteiger	`ops2004amtl/p1ueb21_2004_v10/Umsteiger.txt`
	Sonstiges	· ISO-8859-1 · 4-Spalten-Umsteiger
2.1	URL	`vorgaenger/ops21.zip`
	Kodes	`ops21amtl/p1ueb20_21_v10/opsv21.txt`
	Umsteiger	`ops21amtl/p1ueb20_21_v10/Umsteiger.txt`
	Sonstiges	· ISO-8859-1 · KOMBI-Kode · 6-Spalten-Umsteiger (ursprüngliches Format)
2.0	URL	`vorgaenger/ops20.zip`
	Kodes	`p1ueb11_20_v11/Opsv20.txt`
	Umsteiger	`p1ueb11_20_v11/Umsteiger.txt`
	Sonstiges	· ISO-8859-1 · KOMBI-Kode · 3-Spalten-Umsteiger
1.1	Kodes	`p1ueb11_20_v11/Opsv11.txt`
	Sonstiges	· ISO-8859-1 · Keine Überleitung

A.2 Beispielergebnis der Horizontalen Suche

Ergebnis im JSON-Format für den Aufruf:
`searchHorizontal('icd10gm', '2014', 'M21.6')`

Um Platz zu sparen enthält nur der erste Umsteiger die zusätzlichen Informationen, die pro Eintrag ermittelt werden, also die automatische Überleitbarkeit sowie die Titel der Kodes. Außerdem sind für die Lesbarkeit die Sonderzeichen nicht kodiert.


```
{
  "fwd": {
    "year": "2014",
    "other": "2015",
    "umsteiger": [
      {
        "old": "M21.6",
        "new": "M21.60",
        "auto": "",
        "auto_r": "A",
        "old_name": "Sonstige erworbene Deformitäten des Knöchels und des Fußes",
        "new_name": "Erworbener Hohlfuß [Pes cavus]"
      },
      {
        "old": "M21.6",
        "new": "M21.61",
      },
      {
        "old": "M21.6",
        "new": "M21.62",
      },
      {
        "old": "M21.6",
        "new": "M21.63",
      },
      {
        "old": "M21.6",
        "new": "M21.68",
      }
    ]
  },
```

```
  "rev": {
    "year": "2013",
    "other": "2012",
    "umsteiger": [
      {
        "old": "M21.60",
        "new": "M21.6",
        "recursion": {
          "year": "2.0",
          "other": "1.3",
          "umsteiger": [
            {
              "old": "M21.6",
              "new": "M21.60",
            }
          ]
        }
      },
      {
        "old": "M21.67",
        "new": "M21.6",
        "recursion": {
          "year": "2.0",
          "other": "1.3",
          "umsteiger": [
            {
              "old": "M21.6",
              "new": "M21.67",
            }
          ]
        }
      },
      {
        "old": "M21.87",
        "new": "M21.6",
        "recursion": {
          "year": "2.0",
          "other": "1.3",
          "umsteiger": [
            {
              "old": "M21.8",
              "new": "M21.87",
            }
          ]
        }
      }
    ]
  }
}
```

A.3 Beispiel ConceptMap

Für Abbildung 3.6 ... §

```
<?xml version="1.0"?>
<ConceptMap xmlns="http://hl7.org/fhir">
  <id value="icd10gm_from:2024_to:1.3_target:2024"/>
  <url value="urn:uuid:0192e098-0871-79d5-a4e6-039398aa7803"/>

<group>
  <source value="2018"/>
  <target value="2024"/>
  <element>
    <code value="G83.8"/>
    <target>
      <code value="G83.6"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.8"/>
      <equivalence value="wider"/>
    </target>
  </element>
</group>

<group>
  <source value="2015"/>
  <target value="2024"/>
  <element>
    <code value="G83.80"/>
    <target>
      <code value="G83.5"/>
      <equivalence value="relatedto"/>
    </target>
  </element>
  <element>
    <code value="G83.88"/>
    <target>
      <code value="G83.6"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.8"/>
      <equivalence value="wider"/>
    </target>
  </element>
</group>

<group>
```

```
  <source value="2004"/>
  <target value="2024"/>
  <element>
    <code value="G83.8"/>
    <target>
      <code value="G83.6"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.8"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.5"/>
      <equivalence value="wider"/>
    </target>
  </element>
</group>

<group>
  <source value="1.3"/>
  <target value="2024"/>
  <element>
    <code value="G83.8"/>
    <target>
      <code value="G83.6"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.8"/>
      <equivalence value="wider"/>
    </target>
    <target>
      <code value="G83.5"/>
      <equivalence value="wider"/>
    </target>
  </element>
</group>
</ConceptMap>
```