Fakultät für Wirtschaftswissenschaften
Wismar Business School

Martin Seip

Automatisches Validieren von Meldedaten der EU-Bankenaufsicht

Heft 03/2020

W D P

Wismarer Diskussionspapiere / Wismar Discussion Papers

Die Fakultät für Wirtschaftswissenschaften der Hochschule Wismar, University of Applied Sciences – Technology, Business and Design bietet die Präsenzstudiengänge Betriebswirtschaft, Wirtschaftsinformatik und Wirtschaftsrecht sowie die Fernstudiengänge Betriebswirtschaft, Business Consulting, Business Systems, Facility Management, Quality Management, Sales and Marketing und Wirtschaftsinformatik an. Gegenstand der Ausbildung sind die verschiedenen Aspekte des Wirtschaftens in der Unternehmung, der modernen Verwaltungstätigkeit, der Verbindung von angewandter Informatik und Wirtschaftswissenschaften sowie des Rechts im Bereich der Wirtschaft.
Nähere Informationen zu Studienangebot, Forschung und Ansprechpartnern finden Sie auf unserer Homepage im World Wide Web (WWW): https://www.fww.hs-wismar.de/.

Herausgeber: Prof. Dr. Hans-Eggert Reimers
Fakultät für Wirtschaftswissenschaften
Hochschule Wismar
University of Applied Sciences – Technology, Business and Design
Philipp-Müller-Straße
Postfach 12 10
D – 23966 Wismar
Telefon: ++49/(0)3841/753 7601
Fax: ++49/(0)3841/753 7131
E-Mail: hans-eggert.reimers@hs-wismar.de

Vertrieb: Fakultät für Wirtschaftswissenschaften
Hochschule Wismar
Postfach 12 10
23952 Wismar
Telefon: ++49/(0)3841/753-7468
Fax: ++49/(0) 3841/753-7131
E-Mail: Silvia.Kaetelhoen@hs-wismar.de
Homepage: https://www.fww.hs-wismar.de/

ISSN 1612-0884
ISBN 978-3-948862-00-8
JEL- Klassifikation: C81, G21, G38

Printed in Germany

Dieses Dokument basiert auf der Diplomarbeit des Autors im Studienfach Wirtschaftsinformatik, die im Jahr 2018 an der Hochschule Wismar eingereicht wurde. Sie wurde 2019 mit dem Gottlob-Frege-Preis der Hansestadt Wismar ausgezeichnet.

Um den für Wismarer Diskussionspapiere üblichen Rahmen nicht zu sprengen, beschränkt sich der Verfasser auf die wichtigsten Gedanken und Ergebnisse. Für ein tieferes Verständnis wird auf die vollständige Arbeit verwiesen. Sie ist online veröffentlicht [Seip_DA] und an der Hochschule Wismar einsehbar.

Für die gute Betreuung dankt der Verfasser den Gutachtern der Diplomarbeit:
Herrn Prof. Dr. rer. nat. Jürgen Cleve, Hochschule Wismar
Frau Prof. Dr.-Ing. Sabine Schumann, HAW Hamburg

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AST	Abstract Syntax Tree (abstrakter Syntaxbaum)
ATL	Annotated Table Layout (kommentiertes Tabellenlayout; Excel-Dateien der EBA, die die Zellen einer Taxonomie verdeutlichen)
C#	C sharp (Programmiersprache des .NET-Frameworks)
DPM	Data Point Model; Datenpunktmodell (sowohl als Konzeptbegriff als auch für die MS- Access-Datenbank der EBA verwendet)
EBA	European Banking Authority (europäische Bankaufsichtsbehörde), Sitz in Paris
EBNF	Erweiterte Backus-Naur-Form (Darstellungsweise für formale Sprachen)
ETL	Extract, Transform, Load (Prozess zum Extrahieren, Umformen, Laden von Daten)
FinRep	Financial Reporting (Meldung von Finanzinformationen; untersuchter Meldebereich)
HGB	Handelsgesetzbuch
IFRS	International Financial Reporting Standards (internationale Rechnungslegungsstandards)
LINQ	Language integrated query (C#-Sprachbestandteil)
MS	Microsoft (Softwarehersteller)
MSSQL	Microsoft SQL Server (Datenbankmanagementsystem)
MSVS	Microsoft Visual Studio (Entwicklungsumgebung)
SQL	Structured query language (Abfragesprache für Datenbanken)
SSMS	Microsoft SQL Server Management Studio (Benutzeroberfläche für MSSQL)
UML	Unified Modeling Language (grafische Modellierungssprache)
XBRL	eXtended Business Reporting Language (XML-basiertes Datenaustauschformat)
XML	eXtensible Markup Language (erweiterbare Auszeichnungssprache)

1 Einleitung

Die weltweite Finanzmarktkrise der Jahre 2007ff und ihre Folgen haben die Wirtschaftswelt nachhaltig erschüttert. Aus den Schlagzeilen dieser Zeit sind dem Leser womöglich noch Begriffe wie „Lehman-Brothers-Pleite“, „Subprime-Krise“ und „Regulierungsversagen“ geläufig. Als Reaktion auf die erkannten Ursachen und zur Verhütung einer Wiederholung dieser Krise hat die EU die Aufsicht über die europäische Finanzbranche erheblich verschärft, vgl. [BrSc_BA, S. 593ff.]. Unter anderem sind europäische Banken seitdem zu regelmäßigen, sehr fein detaillierten Meldungen verpflichtet, mit denen die damals neugeschaffene **EU-Bankenaufsichtsbehörde EBA** und die nationalen Aufsichtsbehörden die Finanz- und Risikolage der meldenden Banken besser beurteilen können.

In diesen elektronischen Meldungen (XML-Dateien im XBRL-Format) sind neben den üblichen Messgrößen aus Bilanz, GuV und Anhang[1] zahlreiche Zusatzkennzahlen zu liefern, vgl. Artikel 99 der [EP_VO_575_2013]. Erkennt die Aufsicht aus einer Meldung Handlungsbedarf bei einer Bank, stehen ihr etwa nach Artikel 18 der [EU_VO_1024_2013] Sanktionsmöglichkeiten zur Verfügung. Es liegt somit auf der Hand, dass solche Meldungen an die EBA vollständig und inhaltlich widerspruchsfrei sein müssen.

Durch die oft heterogene IT- und Prozesslandschaft in Banken und die sehr feine Granularität der geforderten Daten ist diese nötige Stimmigkeit der eingereichten Meldungen jedoch oft nicht gegeben. Aus diesem Grund prüft die Aufsicht die eingereichten Meldungen elektronisch mit formelartigen **Validierungsregeln**. Eine typische Regel prüft etwa die Gleichheit von Aktiva und Passiva in der Bilanz. Andere Regeln rechnen Summenpositionen nach, kontrollieren Grenzwerte und Vorzeichen oder stellen einander ausschließende Datenkonstellationen fest. Eine Meldung wird von der EBA nur akzeptiert, wenn sie alle relevanten Validierungsregeln erfüllt; ansonsten wird sie als offensichtlich widersprüchlich zurückgewiesen.

Um valide Meldungen einreichen zu können, sollten Banken ihre Meldedaten vorab hausintern denselben Validierungen unterziehen, die auch die Aufsicht anwendet. Die EBA veröffentlicht die Validierungsregeln und den Meldeumfang auf ihrer Webseite, [EBA_RFW26]. Vorliegendes Werk beschreibt, wie sich diese Daten nutzen lassen, um bankeigene Meldedaten weitgehend automatisiert zu validieren. Durch die häufigen Änderungen an den Melderastern und -regeln wird dieser Prozess immer wieder nötig werden.

---

[1] Der Autor beschränkt den Untersuchungsumfang auf einen Teil der Meldepflichten: auf das Financial Reporting, kurz **FinRep**.

2 Problemstellung und Ziel

2.1 Problemstellung

Der einzureichenden XML-Datei liegt ein (vereinfacht) dreidimensionales Koordinatensystem zugrunde. Jeder Meldewert ist darin einer Zelle zugeordnet, die sich durch die **Koordinaten table (Tabelle), row (Zeile) und column (Spalte)** beschreiben lässt, vgl. [EBA_ATL_Desc, S. 4]. Diese drei Koordinaten erinnern an Tabellenkalkulationsprogramme, und tatsächlich veröffentlicht die Aufsicht Microsoft-Excel-Dateien zur Veranschaulichung der geforderten Meldedaten, vgl. [EBA_ATL_Finrep]. Abbildung 1 verdeutlicht das Koordinatensystem am Beispiel der Bilanzaktiva. Die typische Schreibweise für Adressen im Koordinatensystem ist **F ##.##, r###, c###**, so finden sich die 5 Mio Geldeinheiten in Zelle F 01.01, r010, c010.

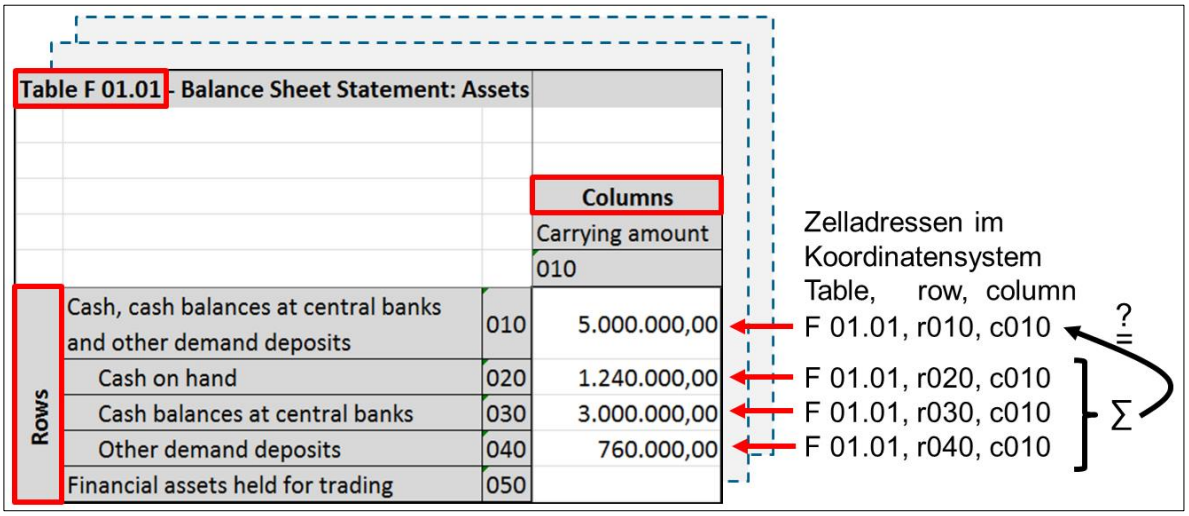


Abbildung 1: Veranschaulichung von Meldedaten am Beispiel von F 01.01 (Aktiva)

Quelle: vereinfacht nach [EBA_ATL_Finrep, Tabelle F 01.01]; Geldbeträge beispielhaft

Durch Texteinrückung wird optisch deutlich, dass die gezeigten 4 Zellen in einer Hierarchie eingeordnet sind. Die 5 Mio Geldeinheiten sollten sich durch Summierung der 3 Unterpositionen ergeben. Wirtschaftlich stellt dies die Detaillierung des Kassenbestands (Zeile 010) in den Bargeldbestand (Zeile 020), Zentralbankguthaben (Zeile 030) und andere täglich fällige Einlagen (Zeile 040) dar. Dieser erwartete Zusammenhang wird durch eine eigene Validierungsregel überwacht. An ihr wird leicht deutlich, wie die erwähnten Koordinaten Zellbereiche eingrenzen, vgl. Abbildung 2:

ID	T1	T2	rows	columns	Formula
v0763_m	F 01.01			(010)	{r010} = sum(r020-040)

Abbildung 2: Einfache Validierungsregel

Quelle: [EBA_VR_XLS, Regel v0763_m]

Deutung dieser Regel: Untersucht wird Tabelle F 01.01 in Spalte 010. In diesem Zellbereich muss die Zeile 010 gleich sein der Summe der Zeilen 020 bis 040. Weitere Tabellen- oder Zeilengrenzen (T2, rows) sind nicht gegeben.

Doch nicht alle Regeln sind so anschaulich. Regel[2] v1031_m in Abbildung 3 untersucht 3 Tabellen in Zeilen- und Spaltenbereichen, verlangt Absolutwerte und spricht nichtzusammenhängende Zellblöcke via xsum an:

ID	T1	T2	T3	rows	columns	Formula
v1031_m	F 31.01	F 07.00	F 43.00			sum({F 31.01, r130, (c010-050)}) <= xsum(abs({F 07.00, (r090-110, r150-180, c080-104)})) + {F 43.00, r070, c050}

Abbildung 3: Komplexe Validierungsregel

Quelle: [EBA_VR_XLS, Regel v1031_m]

Deutung dieser Regel:

- Die Tabellen F 31.01, F 07.00 und F 43.00 werden untersucht (T1 bis T3).
- Spalten und Zeilen dieser Tabellen werden nicht begrenzt (rows und columns sind leer).
- Die Summe der Zellen in Tabelle F 31.01, Zeile 130, Spalten 010 bis 050 bildet die linke Seite des Vergleichs.
- Die rechte Seite wird aus einer Addition (+) gebildet. Deren erster Summand ist die Summe aus den Absolutwerten (abs) von dimensional begrenzten, aber nicht zusammenhängenden Zellen (xsum). Der zweite Summand stammt aus einer einzelnen Zelle.

Eine Veranschaulichung dieser Regel in den Tabellenblättern fällt hier schon schwerer. In Abbildung 4 sind die 3 Terme der Regel farbig markiert und die Zellbezüge in den zugehörigen Farben hervorgehoben.

Ferner ist in Abbildung 4 zu erkennen, dass die Zellen nicht nur durch koordinatenbasierte Adressen – Tabelle, Zeile, Spalte – ansprechbar sind. Jede Zelle hat im Datenmodell der EBA, dem sog. **data point model (DPM)**, vgl. [EBA_AbsDescDPM], eine eindeutige Kennung, z. B. ist Zelle F 31.01, r090, c040 als Zell-ID 39719 im EBA-DPM angelegt. Die Währungssymbole machen deutlich, dass die Zelle eine monetäre Größe darstellt; andere Zellen zeigen Datums-, Ganzzahl- oder Textwerte.

[2] Die EBA vergibt in [EBA_VR_XLS] eindeutige Kennungen für jede Validierungsregel, z. B. v1031_m. Diese Kennungen nutzt auch der Verfasser, um auf Regeln zu verweisen.

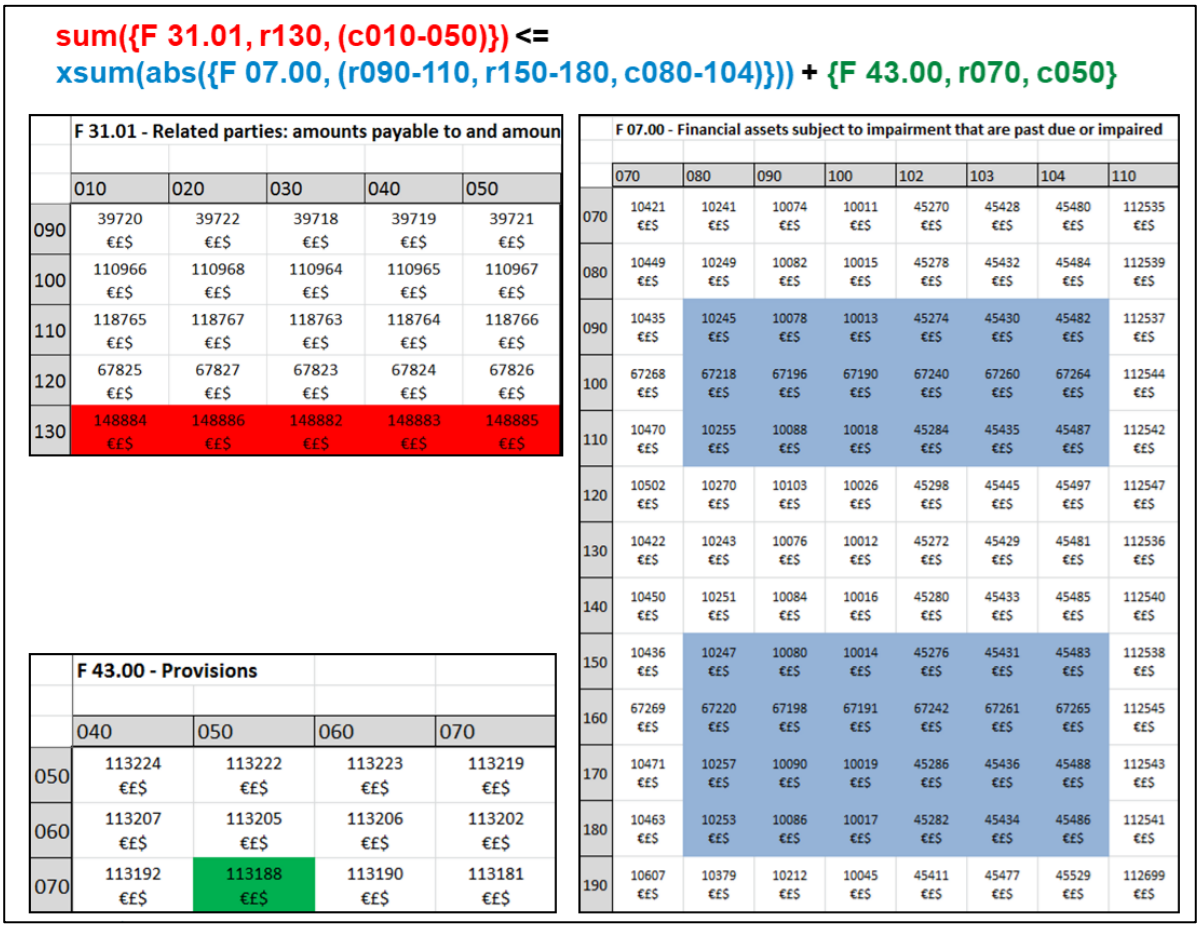


Abbildung 4: Veranschaulichung der Validierungsregel v1031_m im Zellraster

Quelle: oben: Formeltext von v1031_m aus [EBA_VR_XLS]; unten: [EBA_ATL_Finrep, Tabellen F 31.01, F 07.00, F 43.00; Farben vom Verfasser]

Durch die große Anzahl an Regeln wird der Automatisierungsbedarf noch verschärft. Die bisher vorgestellten Regeln sind zwei von Tausenden, wie Abbildung 5 andeutet. Die EBA veröffentlicht die zahlreichen Validierungsregeln in Form einer MS-Excel-Datei, [EBA_VR_XLS]. Deutlich wird, dass bis zu 7 Tabellennamen (T1-T7) auftreten können, sich zahlreiche Änderungsvermerke finden (z. B. „deleted“) und jede Regel einen Typ (type) und einen Schweregrad (severity) aufweist. Der Typ kategorisiert die Regeln, um z. B. Vorzeichenprüfungen („Sign“) von hierarchischen Kontrollrechnungen („Hierarchy“) oder sonstigen Regeln („Manual“) unterscheiden zu können. Der Schweregrad ist relevant bei verletzten Regeln; eine verletzte „blocking“-Regel führt zur Zurückweisung der gesamten Meldung, während verletzte „non-blocking“-Regeln Erläuterungen erfordern oder Rückfragen nach sich ziehen. Diese Angabe wird später ausgewertet, um Meldedaten insgesamt als meldereif, erklärungsbedürftig oder unstimmig zu bewerten.

ID	Replaces	Changed in framework release	Last Change	Deactivated on	Reactivated on	Deleted	Not Implemented in XBRL	Type	Severity	T1	T2	T3	T4	T5	T6	T7	rows	columns	sheets	Formula
v2836_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.02.b							(r250)			{c020} <= {c010}
v2837_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.02.b							(r250)			{c030} <= {c010}
v2838_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.03.a								(c010)		{r010} = sum(r020-030)
v2839_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.03.b							(r010-r060)			{c030} <= {c020}
v2840_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.03.b								(c020-c040)		{r010} = sum(r020-030)
v2841_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.03.b								(c020-c040)		{r030} = sum(r040-060)
v2842_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a							(r010-r050;r070;r090			{c020} <= {c010}
v2843_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010-c020)		{r030} <= {r020}
v2844_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010)		{r060} <= {r050}
v2845_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010)		{r080} <= {r070}
v2846_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010-c020)		{r100} <= {r090}
v2847_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010-c020)		{r110} <= {r090}
v2848_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010-c020)		{r040} = {r050} + {r070}
v2849_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010-c020)		{r010} = {r020} + {r040}
v2850_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010-c020)		{r120} = sum(r130-160)
v2851_m		2.1 (2014/03)	Add					Manual	Blocking	F 32.04.a								(c010-c020)		{r170} = {r010} + {r120}
v2852_m		2.1 (2014/03)	Add				y	Manual	Blocking	F 32.04.a	F 01.02									{F 32.04.a, r020,c010} <

Abbildung 5: Validierungsregeln im MS-Excel-Format der EBA (Auszug)

Quelle: [EBA_VR_XLS, Regel v2836_m ff]

2.2 Ziel

Der Autor stellt sich die Aufgabe, die ca. 2800 Validierungsregeln maschinell so aufzubereiten, dass sich die jeweils angesprochenen Zellen und die verlangten Berechnungen und Vergleiche automatisch ableiten und auf Meldedaten anwenden lassen.

Die entstehende Anwendung *par*st und *k*alkuliert somit *E*BA-*Val*idierungsregeln. Mit etwas Nachsicht in der Rechtschreibung und Rücksicht auf die Artussage leitet sich so der Programmname **Parceval** für die entstehende Anwendung ab. Parceval besteht aus einer MS-SQL-Server-Datenbank als Datenhaltungsschicht und einer in MS Visual Studio entwickelten C#-Solution für Benutzeroberfläche und Programmlogik.

3 Datenbeschaffung und Parsergenerierung

Die Problemlösung umfasst folgende Schritte, die im weiteren Verlauf des Kapitels näher beschrieben werden:

- Daten aus den Vorgaben der EBA (DPM-Zellen, Validierungsregeln) werden in eine Datenbank eingelesen.
- Die Bestandteile der Validierungsregeln werden durch formale Grammatiken beschrieben. Für die Formel liegt eine Grammatik der EBA vor, für die Basiskriterien werden sie vom Verfasser aufgebaut.
- Aus diesen Grammatiken werden mittels eines frei verfügbaren Parsergenerators Analyseprogramme (Parser) in C# erstellt.
- Diese Parser werden in der entwickelten Anwendung Parceval auf die DPM-Zellen und Validierungsregeln angewendet.
- Ergebnis sind maschinell auswertbare Zusammenhänge von Zell-Platzhaltern, die in einer Datenbank gespeichert werden.
- Eine Validierung von Meldedaten geschieht, indem Parceval die Zell-Platzhalter der Regeln mit den Werten des zu prüfenden Meldebestands belegt und anschließend die in den Regeln verlangten Operationen ausführt. Jeder Regel evaluiert dabei für die betrachtete Meldung zu „erfüllt“ oder „verletzt“. Sind alle Regeln erfüllt, ist die Meldung valide und damit meldereif. Alle Ergebnisse sind am Bildschirm nachvollziehbar, was eine evtl. Ursachensuche in Zuliefersystemen erleichtert.
- Ein Generator und Editor für Meldedaten erlaubt umfangreiche Tests von Parceval, ohne Meldedaten aus dem Produktionsbetrieb vorauszusetzen.

3.1 Datenbeschaffung

3.1.1 Validierungsregeln

Die MS-Excel-Datei der Validierungsregeln, [EBA_VR_XLS], wird von der EBA-Webseite heruntergeladen. Die Datei enthält Regeln mehrerer DPM-Versionen; für den Untersuchungsumfang ist Version 2.6 relevant. Dieses Blatt wird durch manuellen Import in eine Datenbanktabelle von Parceval eingelesen. Dabei ist sorgfältig auf Wahl und Wandlung der Datentypen, hinreichende Feldlängen und korrekte Verarbeitung der mehrzeiligen Überschriften zu achten. Regeln, die von der EBA im Änderungsprotokoll als „deleted“ (gelöscht) oder „deactivated“ (deaktiviert) markiert sind, werden entfernt, da sie nicht mehr anzuwenden sind. Ergebnis sind in der vorliegenden Untersuchung der DPM-Version 2.6 **2849 Regel-Datensätze**.

3.1.2 Zellen des DPM („Melderaster“)

Die MS-Excel-Dateien, die das Melderaster veranschaulichen, vgl. Abbildung 4, eignen sich nicht zum Import in relationale Datenbanken. Stattdessen nutzt der Verfasser die MS-Access-Datei der EBA [EBA_DPMDBv2.6.0.0], um die Zellen des Untersuchungsumfangs zu isolieren. Durch Analyse des Beziehungsgeflechts und mit Hilfe der DPM-Dokumentation der EBA in [EBA_DPMDB_Desc_v2.0.0] leitet er einige Abfragen ab, die die FinRep-Zellen der DPM-Version 2.6 liefern. Das Endergebnis wird ebenfalls manuell in die Parceval-Datenbank importiert.

3.2 These: Validierungsregeln als Worte formaler Sprachen

Bei genauerer Untersuchung der Validierungsregeln fällt auf, dass jede aus mehreren Teilen besteht. Abbildung 6 stellt diese Teile vor.

- **Basiskriterien**
  Der Anwendungsbereich der Regel wird durch Tabellen-, Zeilen- und/oder Spaltenkriterien eingegrenzt. Nur Zellen, die allen Basiskriterien gemeinsam genügen, vom Verfasser **Basiszellen** genannt, werden in der Formel der Validierungsregel weiter untersucht. Im Bild sind aus Platzgründen nur die Tabellen T1-T3 gezeigt; es sind insgesamt bis zu 7 Tabellen (T1-T7) in den Basiskriterien möglich.
- **Formel**
  Die Formel bildet einen Vergleichsausdruck, der auf die Basiszellen angewendet wird. Jeder Term der Formel beschreibt über seine Kriterien (Tabelle, Zeile, Spalte) eine Teilmenge der Basiszellen und kann diese mit Funktionen verrechnen. Die gesamte Formel verrechnet diese Terme, etwa durch Addition (+) oder Subtraktion (-), und bildet durch eine Vergleichsoperation einen Wahrheitswert. Beispiel: „wenn Wert aus Zelle X plus Wert aus Zelle Y gleich Wert aus Zelle Z, dann ist die Regel erfüllt“.

	Basiskriterien (begrenzen Basis)					Formel (auf Basis anzuwenden)
ID	T1	T2	T3	rows	columns	Formula
v0978_m	F 20.02	F 01.02				{F 20.02, r210, c010} + {F 20.02, r210, c020} = {F 01.02, r290, c010}
v0979_m	F 20.02	F 01.02				{F 20.02, r220, c010} + {F 20.02, r220, c020} = {F 01.02, r300, c010}
v0980_m	F 20.03				(010-020)	{r250} = {r010} - {r020} - {r030} + {r040} + {r050} - {r060} + {r070} + {r080}
v0985_m	F 20.04				(010)	{r010} >= sum(r020-030)
v0986_m	F 20.04				(010)	{r040} >= sum(r050-070)
v0987_m	F 20.04				(010-030)	{r080} = sum(r090-130)
v0988_m	F 20.04				(010-030)	{r140} = {r150} + {r160} + {r170} + {r180} + {r190} + {r220}
v0992_m	F 20.05.a	F 20.05.b		(010-030)		{F 20.05.a, c010} >= {F 20.05.b, c030}

Abbildung 6: Bestandteile von Validierungsregeln

Quelle: [EBA_VR_XLS]; Ausblendung unnötiger Details sowie Farbkennzeichnung durch den Verfasser

Diese Regelbestandteile sind nicht völlig beliebig gestaltet. Schon mit bloßem Auge fällt auf, dass die bestehenden Regeln ein gewisses formales Schema einhalten. Die Formeln sind letztlich Ausdrücke, die zwei Seiten einem Vergleich unterziehen, und auch die Basiskriterien scheinen aus Strukturen wie Einzelwert (z. B. „F 20.04“) oder Bereich (z. B. „010-030“) zu bestehen.

Dies führt zu folgender These:

Aufgrund ihrer formalen Struktur lassen sich die 4 Bestandteile jeder **Validierungsregel als Worte formaler Sprachen** im Sinne der theoretischen Informatik darstellen. Diese formalen Sprachen lassen sich jeweils über eine **formale Grammatik** beschreiben, die in Form von Ersetzungsregeln mögliche Entwicklungsschritte vom Ursprungs- zum Zieltext aufführt. Aus den 4 Grammatiken erstellt ein **Parsergenerator** jeweils 4 C#-Klassen, deren Einsatz aus den Regeltexten **Parsebäume** erzeugt – verschachtelte Abfolgen von Grammatikregeln, die die innere Struktur des Textes offenlegen, vgl. [WaHi_FSAAC, S. 31]. Diese Parsebäume werden dann von Parceval untersucht, um relevante Teile wie Suchkriterien, Vergleichs- und Rechenanweisungen zu erkennen und daraus ein **Regelskelett** in Form eines **abstrakten Syntaxbaums** zu erzeugen. Seine Anwendung auf Meldedaten beurteilt deren Meldereife und kann so Fehlmeldungen vermeiden helfen.

Der Parser für die Basiszeilentexte ist somit in der Lage, aus dem Beispieltext „010;130-480“ die Struktur „Zeile 010 und außerdem Zeilen 130 bis 480“ zu erkennen. Diese Struktur nutzt Parceval später als Suchkriterium für die passenden Zeilen. Abbildung 7 zeigt den Parsebaum für diese Basiszeilenrestriktion und hebt hervor, welche Teile Kriterien bilden.

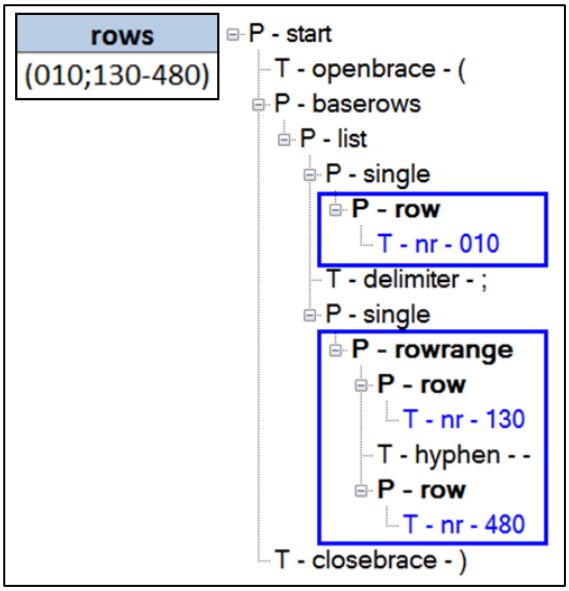


Abbildung 7: Parsebaum der Basiszeilenrestriktion (010;130-480)

Quelle: Eigenes Werk aus Bildschirmfoto von Parceval

Analog dazu ist der Formelparser in der Lage, die Formel in einen Parsebaum zu übersetzen. Die entstehende Baumstruktur bildet auch die Rangfolge der Operationen ab, die der Formeltext u. a. durch Textrichtung, Klammern und ggf. impliziten Operatorvorrang (Punkt vor Strich) ausdrückt. Da die Formelgrammatik – anders als die Grammatiken der 3 Basiskriterien – rekursiv absteigt, ist die Tiefe des Baums nicht begrenzt und der Parsebaum nicht „flach“. Abbildung 8 zeigt einen Ausschnitt aus einem Formel-Parsebaum und hebt relevante Bereiche hervor. Deutlich wird, dass der Vergleichsoperator gleichsam als Scharnier zwischen den beiden Seiten links (L) und rechts (R) des Vergleichsausdrucks liegt.

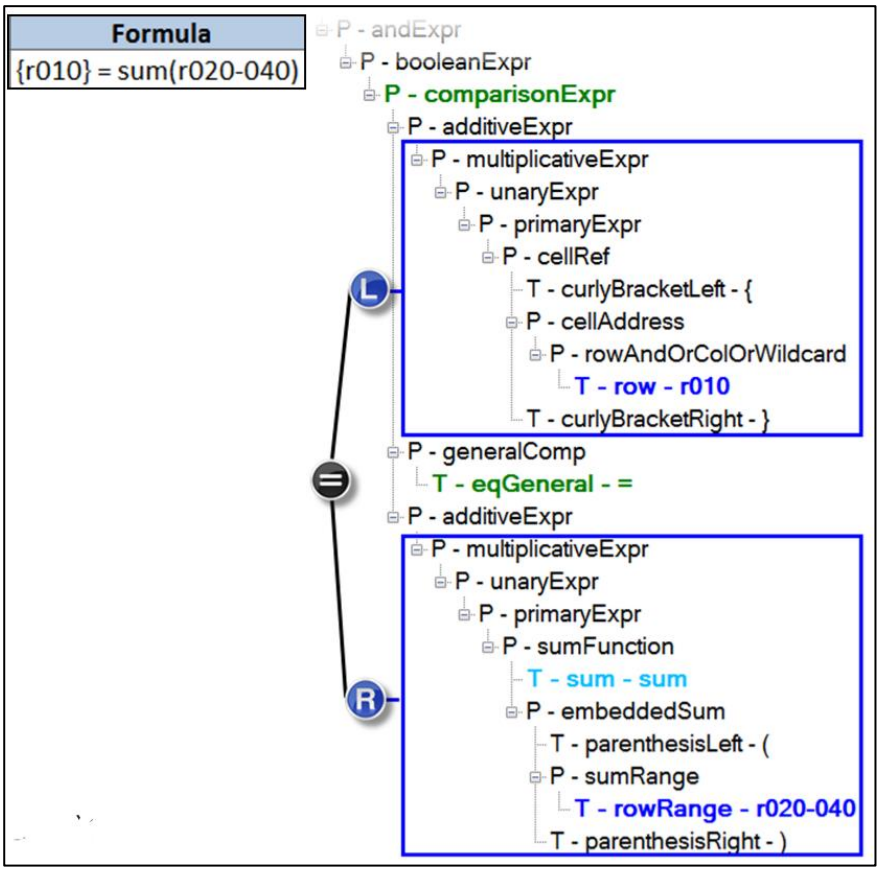


Abbildung 8: Parsebaum einer Formel (Auszug)

Quelle: Eigenes Werk aus Bildschirmfoto von Parceval

3.3 Grammatiken der Regelbestandteile

Für die Formeln der Validierungsregeln liefert die EBA eine formale Grammatik in den „EBA Taxonomy files and supporting documentation“ [EBA_TaxDocZIP]. Diese formale Grammatik enthält 55 Tokens (Terminale, ergeben Blätter im Parsebaum) und 46 Produktionen (Nonterminale, ergeben

innere Knoten im Parsebaum). Sie ist im EBNF[3]-nahen Format des **Parsergenerators Grammatica** [Cederberg_Grammatica] verfasst und als Textdatei „Validation Formula.grammar“ mit jedem Editor lesbar. Abbildung 9 gewährt einen Einblick in diese Grammatik. Erkennbar ist, dass jeder Eintrag eine Ersetzung der linken Seite durch die rechte Seite ermöglicht und die Namen einiger Regeln (links) in den Ersetzungen (rechts) auftreten. Das Pipe-Zeichen (|) steht für Alternativen.

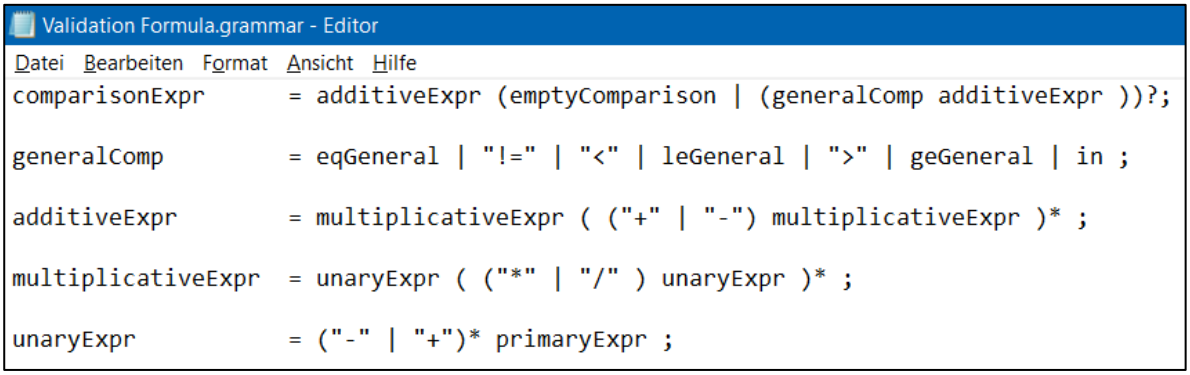


Abbildung 9: EBA-Grammatik der Formeln (Auszug)

Quelle: Datei Validation Formula.grammar aus [EBA_TaxDocZIP] in Windows Editor

Für die Basistabellen, -zeilen und -spalten erstellt der Verfasser eigene Grammatiken. Ausgangspunkt ist die Analyse der Basiskriterien der Regeln, die in die Parceval-Datenbank importiert wurden. Im weiteren Verlauf steht aber auch die EBA-Formelgrammatik Pate, denn auch in den Formeln kommen die für Basiskriterien typischen Strukturen wie Zeilenbereich oder Spaltenliste vor; sie sind daher leicht aus der Formelgrammatik zu isolieren.

Für die **Basiszeilen** entsteht so folgende Grammatik:

```
%header%
// A Grammatica Grammar file - see http://grammatica.percederberg.net/
GRAMMARTYPE    = "LL"
CASESENSITIVE  = "no"
DESCRIPTION    = "row restrictions in EBA validation rules"
AUTHOR         = "Martin Seip"
VERSION        = "2.6"

////////////////////////////////////
%tokens%
////////////////////////////////////
prefix         = "r"%ignore%
delimiter      = ";"
nr             = <<[0-9]?[0-9][0-9][0-9]>> //3 oder 4 Ziffern (4 nur in Corep)
openbrace      = "("
closebrace     = ")"
hyphen         = "-"
allstring      = "(All)"                   //keine einschränkende Wirkung
```

[3] EBNF: Erweiterte Backus-Naur-Form; Notationsweise für formale Sprachen.

```
////////////////////////////////////
%productions%
////////////////////////////////////
start          = allstring
               | baserows
               | openbrace baserows closebrace;
baserows       = single
               | list;
single         = row
               | rowrange;
row            = prefix? nr;
rowrange       = row hyphen row;
list           = (single delimiter)+ single;
```

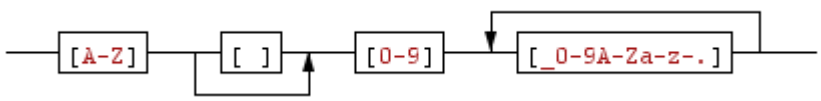
Abbildung 10: Grammatik der Basiszeilen

Quelle: eigenes Werk in der zugrundeliegenden Abschlussarbeit

Deutung: Basiszeilentexte bestehen entweder aus dem Text (All) (*allstring*) oder einer optional eingeklammerten *baserows*-Angabe, die aus einem *single*- oder *list*-Nonterminal besteht. Eine *single*-Produktion entwickelt sich zu einer *row*- oder einer *rowrange*-Produktion weiter. Eine *row* ist eine *nr* mit optionalem *prefix*; eine *rowrange* verbindet zwei solcher *rows* mit einem *hyphen*. Eine *list* ist eine *delimiter*-getrennte Abfolge von mindestens zwei *singles*.

Für die **Basisspalten** ist die Grammatik fast identisch; das *prefix*-Token der Spalten ist c (für column) statt r (für row). Zur besseren Nachvollziehbarkeit werden aber auch die inhaltlich unveränderten Produktionen für Einzelwerte, Bereiche und Listen passend umbenannt (*rowrange* zu *columnrange* etc.).

Für die **Basistabellen** schließlich ist die Grammatik ebenfalls recht einfach. Da jede T1-T7-Angabe nur einen Tabellennamen bezeichnet (oder leer ist), und die Tabellennamen bereits in der EBA-Formelgrammatik als regulärer Ausdruck strukturiert sind, entsteht folgende Grammatik:

Typ	Definition und Syntaxdiagramm
Token	`openbrace = "("` 
Token	`closebrace = ")"` 
Token	`table = <<[A-Z][ ]?[0-9][_0-9A-Za-z-\.]+>>` 

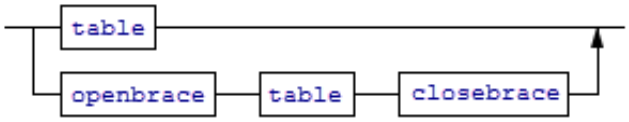
Produktion	start = table | openbrace table closebrace; 

Abbildung 11: Grammatik der Basistabellen

Quelle: eigenes Werk in der zugrundeliegenden Abschlussarbeit

3.4 Parserbau mittels Parsergenerator

Die Grammatiken beschreiben zwar, wie korrekte Texte der Validierungsregeln aufgebaut sein müssen, können aber vorhandene Regeltexte nicht untersuchen. Hierfür werden **Parser** benötigt – spezielle Programme, die die Elemente einer Grammatik im untersuchten Text wiedererkennen und die Ersetzungsregeln der Grammatik so anwenden, dass der Text schrittweise abgeleitet werden kann, vgl. [VoWi_GTI, S. 226]. Ein Parser erzeugt somit einen **Parsebaum** als Pfad zum Ziel – die Abfolge anzuwendender Ersetzungsregeln der Grammatik. Der Parsebaum offenbart die Struktur des Inhalts, im vorliegenden Anwendungsfall etwa Vergleiche, Summen oder Zeilenbereiche.

Im einfachsten Fall testet ein Parser von der Startproduktion ausgehend schrittweise alle Möglichkeiten, die ihm die Grammatik bietet, um den Text nach und nach aufzubauen. Jede anwendbare Produktion ist eine Wegscheide, an der sich neue Fortschreitemöglichkeiten bieten – darunter womöglich die richtige Regelabfolge, aber auch Irrwege, die den Text letztlich doch nicht erzeugen. Damit ähnelt das Parsen eines Textes der Routensuche eines Navigationssystems, das von einem Start (Startproduktion) zu einem Ziel (abzuleitender Text) auf einem gegebenen Wegenetz (Grammatik) die Route (Parsebaum) sucht.

Das systematische Suchen nach der Textstruktur in eine Programmstruktur zu codieren klingt aufwendig, kann jedoch im vorliegenden Fall an einen **Parsergenerator** delegiert werden. Parsergeneratoren sind Anwendungen zur automatisierten Erzeugung von Parsern. Sie „nehmen die Beschreibung der Quellsprache auf und erzeugen ein zugehöriges Analyseprogramm für diese Sprache.“ [WaHi_FSAAC, S. 123] Die Sprachbeschreibung wird ihnen als Grammatik in strukturierter Form, etwa in EBNF, übergeben.

Parsergeneratoren übernehmen dann die Generierung eines Parsers, der zur Laufzeit die Verzweigungen (Produktionen) und Endpunkte (Tokens) der Grammatik systematisch durchsucht, um den Parsebaum aufzubauen. Weite Verbreitung hat der Parsergenerator ANTLR [Parr_ANTLR] gefunden, doch die EBA hat sich bei der Formelgrammatik stattdessen für Grammatica [Cederberg_Grammatica] entschieden, so dass auch der Autor diesem Entschluss folgt.

Grammatica wird als kommandozeilengesteuertes Java-Archiv (*.jar) ausgeliefert. Durch Aufruf von Grammatica mit einem der gut dokumentierten Parameter aus [Cederberg_CLI] entsteht aus der übergebenen Grammatik C#-Code, wie Abbildung 12 zeigt. Es entstehen 4 C#-Dateien je Grammatik, die sich in Entwicklungsumgebungen wie Microsoft Visual Studio (MSVS) einfügen lassen. Insgesamt geschieht dieser Prozess für alle 4 Grammatiken und erzeugt somit 16 Dateien für die Parceval-C#-Projektmappe in MSVS.

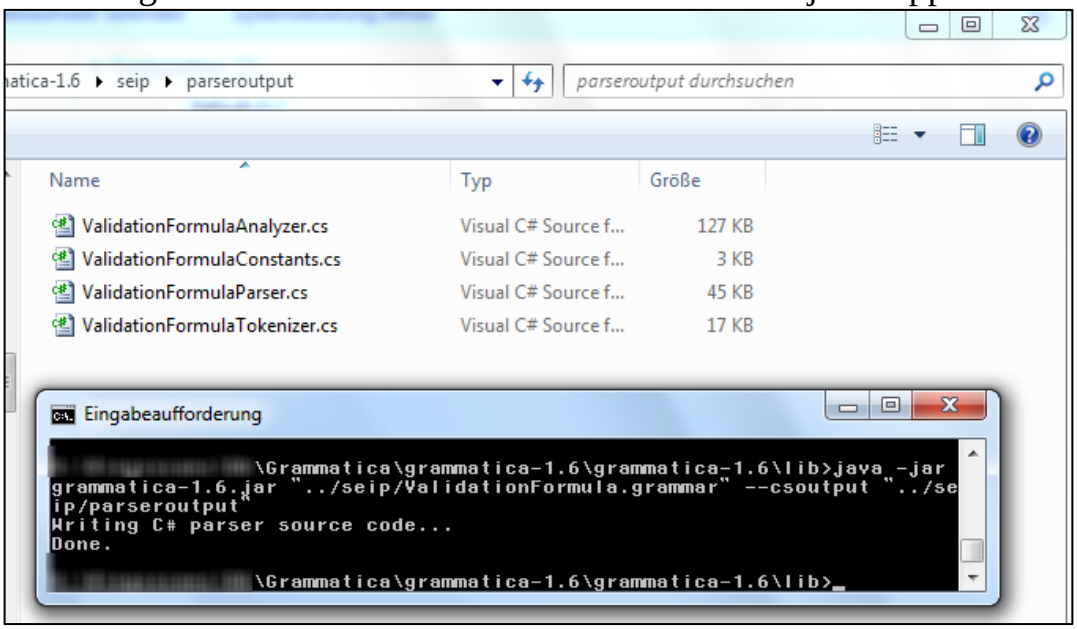


Abbildung 12: Parsergenerierung mit Grammatica

Quelle: Eigenes Werk; Bildschirmfoto des Java-Aufrufs von Grammatica in der DOS-Eingabeaufforderung (vorn) und Ergebnisse im Windows Explorer (hinten)

3.5 Einsatz der generierten Parser

Nach Einbinden der 16 C#-Dateien in die Parceval-Projektmappe in MSVS leitet der Verfasser die 4 generierten Analyzer-Klassen manuell zu **Inspector-Klassen** ab, um den generierten Code nicht abändern zu müssen. Diese Inspector-Klassen erhalten in ihrer Inspect()-Methode den zu untersuchenden Text übergeben, parsen ihn mit Hilfe der von Grammatica generierten

Analyzer-Klassen und geben, falls der Text grammatikgemäß ist, einen Verweis auf den Wurzelknoten des Parsebaums zurück, d. h. auf die Startproduktion. In den grammatica-generierten Analyzer-Klassen sind für *alle* Grammatikregeln override-Methoden vorgesehen. In den abgeleiteten Inspector-Klassen werden davon jedoch nur diejenigen implementiert, die für Parceval *relevant* sind. Abbildung 13 zeigt die Klassendiagramme für die 3 Inspector-Klassen der Basisrestriktionen. Die jeweils nicht implementierten Methoden der Analyzer-Klassen stehen für irrelevante Bestandteile, die keine Auswirkung auf die zu ermittelnden Daten haben. So sind im generierten BaseRowsAnalyzer etwa die Methoden EnterDelimiter() und ExitDelimiter() vorgesehen, sie werden aber im BaseRowsInspector nicht implementiert, da der delimiter selbst keine Relevanz hat – er trennt lediglich single-Produktionen voneinander, die von ExitSingle() verarbeitet werden.

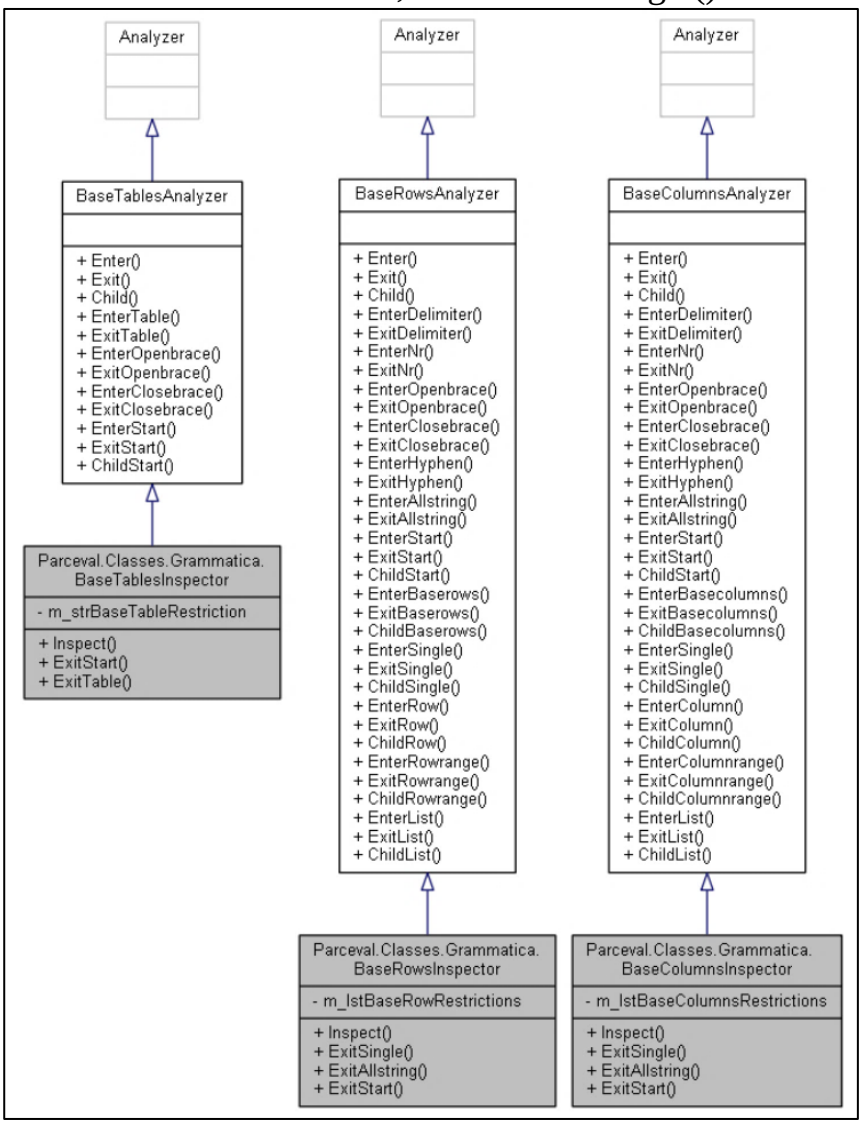


Abbildung 13: Inspector-Klassen der Basisrestriktionen

Quelle: UML-Klassendiagramme der Klassen BaseTablesInspector, BaseRowsInspector und BaseColumsInspector, erstellt via Doxygen aus dem Parceval-Quellcode

Je nach Struktur des Textes kann der entstehende Parsebaum aus sehr vielen Knoten bestehen und/oder sehr tief verschachtelt sein. Jeder **Knoten im Parsebaum** ist eine Instanz der Klasse Node (vgl. Abbildung 14), die Grammatica bereitstellt, um die Informationen eines Ableitungsschritts im Parsebaum darzustellen. Für jeden Node ist so z. B. ersichtlich der *Name* und die *ID* der angewandten Grammatikregel, der übergeordnete Node (*Parent*) sowie die Position im Eingabetext, an der die Regel angewendet wurde (*StartLine*, *-Column*, *EndLine*, *-Column*). Abbildung 15 zeigt das Ergebnis einer erfolgreichen Ableitung eines Basiszeilentextes durch den BaseRowsInspector im Überwachungsfenster der Entwicklungsumgebung. Die start-Produktion steht als Wurzel des Baums ganz oben, die weiteren Ableitungsschritte sind als verschachtelte Node-Instanzen erkennbar.

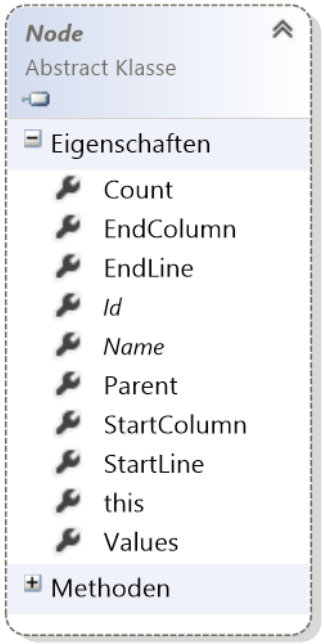


Abbildung 14: Klasse Node

Quelle: Klassendiagramm der Klasse Node aus MSVS

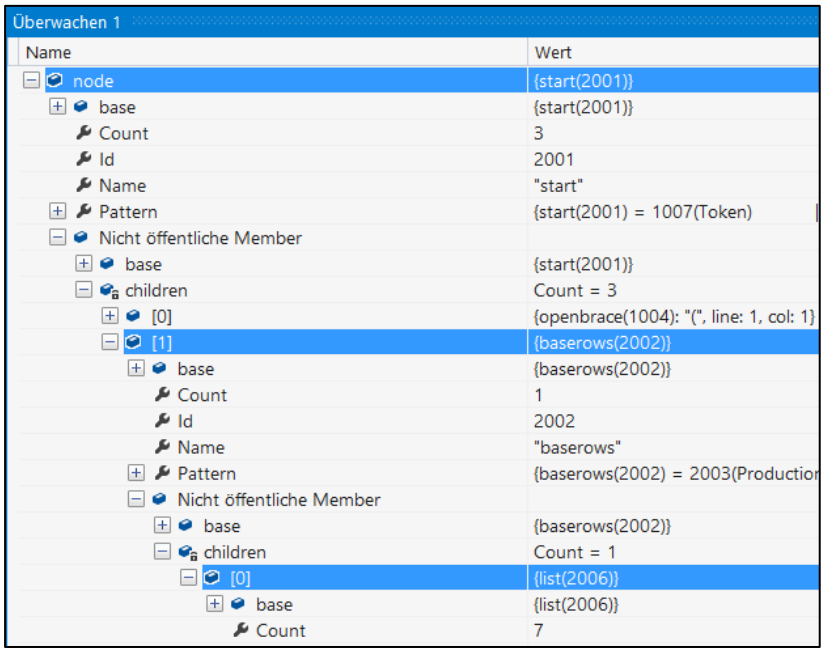


Abbildung 15: Parsebaum (Auszug)

Quelle: Debugger-Überwachungsfenster der Projektmappe Parceval; Eingabetext ist die Basiszeilenrestriktion (170;190;210-220;250-270)

Lässt sich der abzuleitende Text nicht aus den Grammatikregeln bilden, ist er nicht grammatikgemäß. Es gibt in diesem Fall keinen Parsebaum, und der Text ist formal falsch. Überraschenderweise ist dieser Fall durchaus nicht selten. In den 2849 Regeln des Untersuchungsumfangs gab es eine Regel, die gegen die Basistabellen-Grammatik verstieß und 397 Regeln (sic!), deren **Formel grammatisch falsch** war (Details dazu in der Abschlussarbeit). Die EBA verstößt also mehrfach gegen die Formelgrammatik, die sie sich selbst auferlegt hat. Die betroffenen Regeln markiert Parceval unter Nennung von Fehlertext und -position als nicht deutbar und verarbeitet sie nicht weiter.

4 Auswerten der Parsebäume

In Parceval werden die Parsebäume als TreeViews veranschaulicht. Für jede Regel lassen sich die 4 Parsebäume direkt im Programm anzeigen, wie Abbildung 16 illustriert. Die Basisspaltenrestriktion (120-160) oben im Bild erzeugt einen Parsebaum von 11 Knoten Länge und 6 Ebenen Tiefe. Parceval erkennt die Eingrenzung (*single*) als Spaltenbereich (*columnrange*) von zwei Spalten (*column*). Werden Äste im TreeView markiert, hebt Parceval den zugehörigen Textbereich hervor.

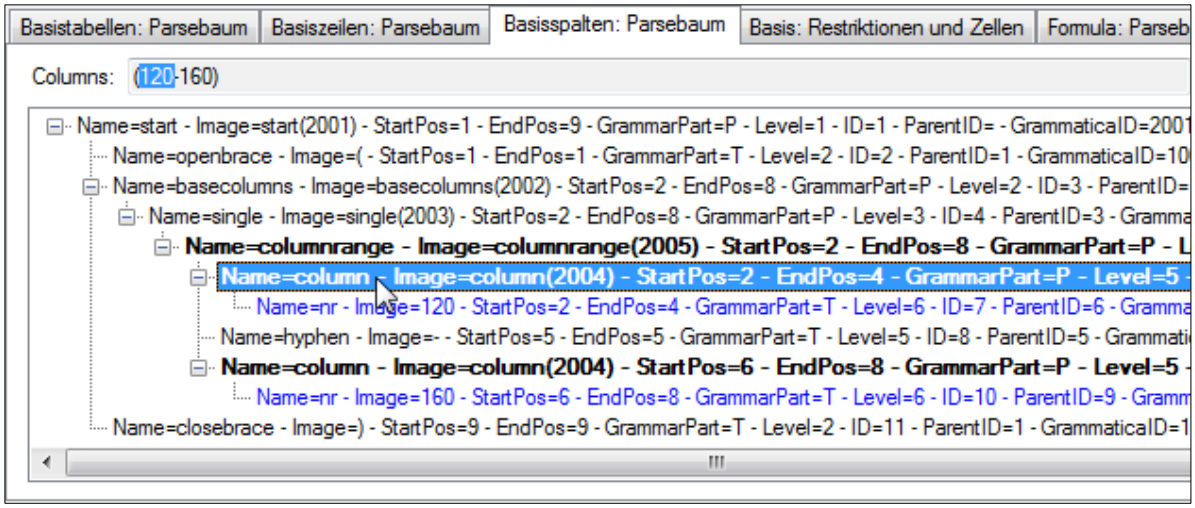


Abbildung 16: Parsebaum eines Basiszeilentextes als TreeView

Quelle: eigenes Werk, Bildschirmfoto aus Parceval

Wie sich zeigt, sind Parsebäume schon für kurze Texte recht umfangreich. Für längere und komplexere Texte explodiert ihre Größe regelrecht[4].

Zudem enthalten sie viele Knoten, die nur grammatisch bedingte Zwischenschritte sind und keine Auswirkung auf das Endergebnis haben. So kommt etwa das *hyphen*-Token in den Grammatiken der Basisrestriktionen nur als Trenner der Grenzen in Bereichen vor, die sich schon zuvor durch ihre *range*-Produktion ankündigen. Parr nennt Parsebäume in [Parr_LIP, S. 88] „*full of noise*“ und „*extremely inconvenient to walk and transform*“. Um dieses grammatische Rauschen („**noise**”) von den Nutzdaten zu scheiden, ist eine tiefere Analyse nötig.

[4] Extremfälle sind die Parsebäume der Formeln der Regeln v1141_m (780 Knoten lang) und v4568_m (36 Ebenen tief).

Die Kernidee dieser Analyse soll hier zunächst in aller Kürze verdeutlicht werden, Details folgen in den Unterkapiteln:

- Die Parsebäume der 4 Inspector-Klassen werden in DataTables **serialisiert**, um sie der leistungsstarken Abfragesprache **LINQ** in C# zugänglich zu machen.
- Aus den DataTables der Parsebäume der drei **Basisrestriktionen** werden per LINQ die **Suchkriterien isoliert**. Die entstehenden DataTables begrenzen den Gesamtvorrat der DPM-Zellen auf die **Basiszellen** der Validierungsregel, die die Formel untersucht.
- In der DataTable des **Formel**-Parsebaums wird der Einstiegspunkt gesucht, an dem der **Vergleich** beginnt[5]. Dessen Nachfolger im Baum werden systematisch auf **arithmetische und dimensionale Bestandteile** durchsucht. Die arithmetischen Bestandteile bilden die Rechenterme im **Regelskelett** des verlangten Vergleichs. Die dimensionalen Bestandteile der Formel ermitteln für jeden so erkannten Term diejenigen Basiszellen, die er verrechnet.
- Das Regelskelett und die Zellen, die dessen Terme verarbeiten, werden in einer **Datenbank** abgelegt. Dort stehen sie für das Validieren einer Meldung zur Verfügung.

Abbildung 17 und Abbildung 18 illustrieren diese Kernidee. Erkennbar ist das verschachtelte Teilmengenprinzip der dimensionsbasierten Suchkriterien, das sich in Basis- und Formelbereich wiederholt (Abbildung 17), sowie der schrittweise Aufbau eines Regelskeletts aus der Formel (Abbildung 18).

---

[5] Die Wahl dieses Einstiegspunkts – er ist nicht der Wurzelknoten des Parsebaums – schließt einige Regeln von der Verarbeitung durch Parceval aus. Dies sollte in künftiger Forschung erarbeitet werden.

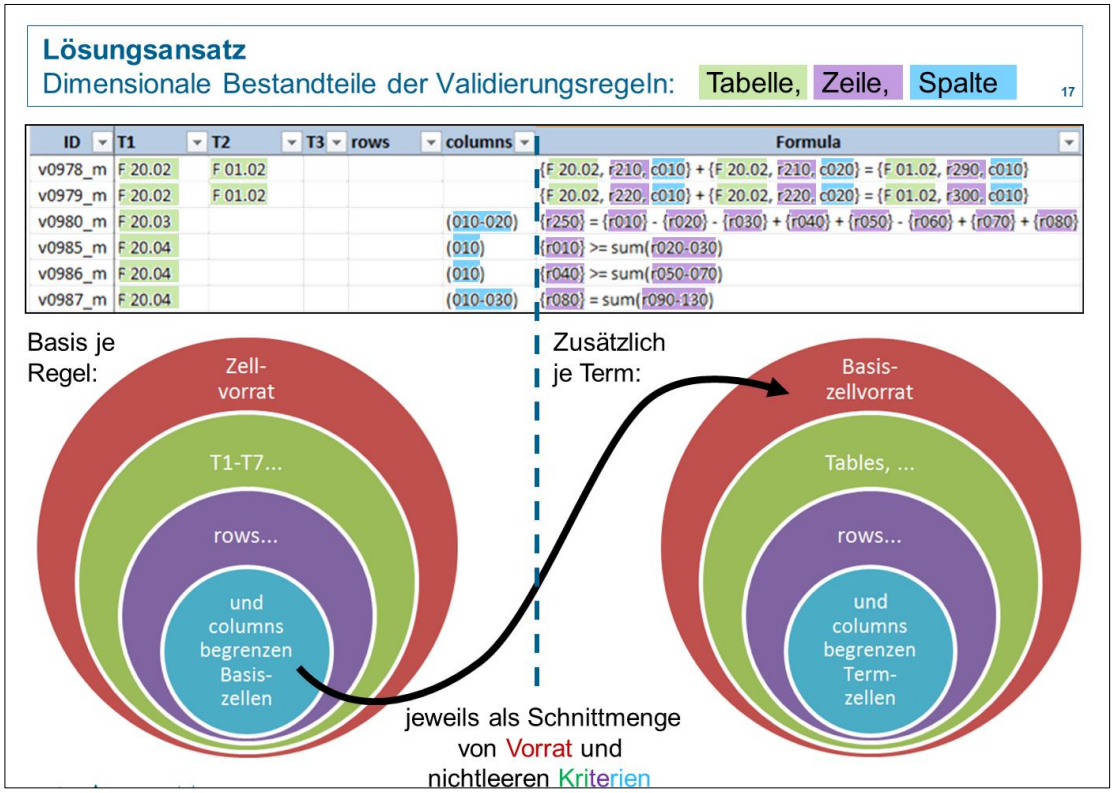


Abbildung 17: Dimensionale Bestandteile von Validierungsregeln

Quelle: eigenes Werk unter Nutzung einiger Regeln aus [EBA_VR_XLS]

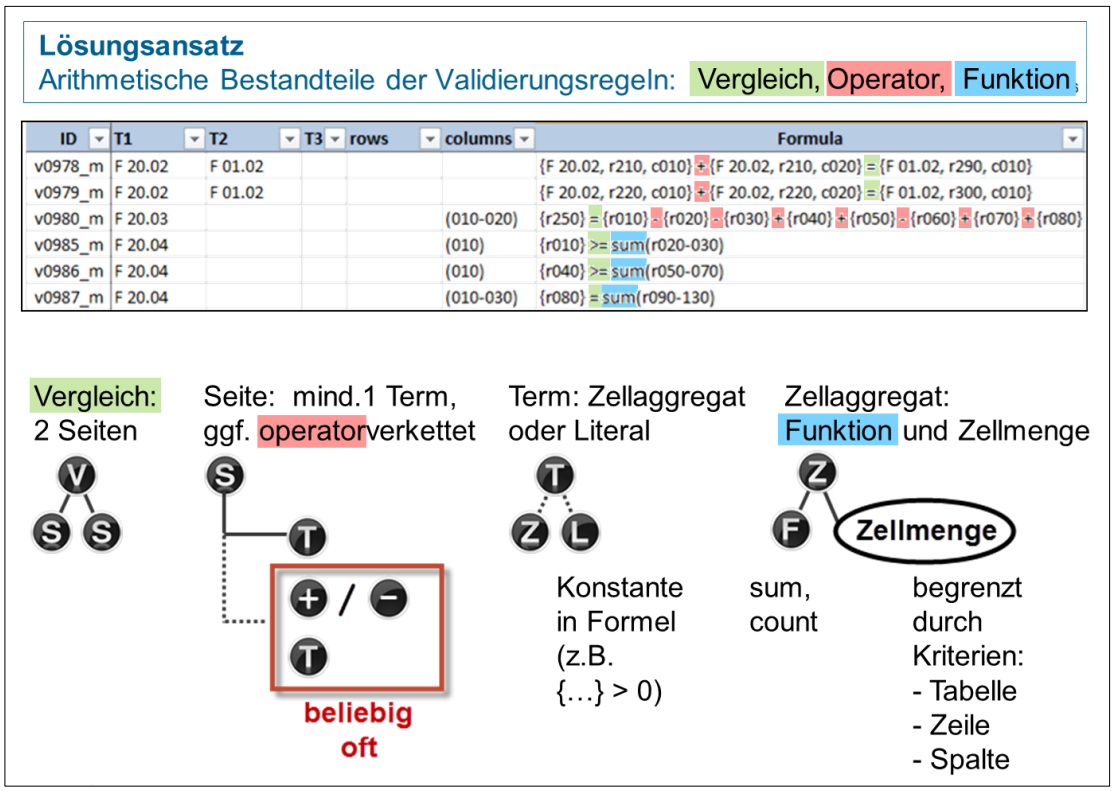


Abbildung 18: Arithmetische Bestandteile von Validierungsregeln

Quelle: eigenes Werk unter Nutzung einiger Regeln aus [EBA_VR_XLS]

Der Verfasser erstellt für die Analyse der Parsebäume zwei Klassen:

- clsParser führt die nötigen Schritte zum Parsen *einer* Validierungsregel aus.
- clsParserController koordiniert einen Parselauf für *alle* Validierungsregeln. Diese Klasse erzeugt für jede zu verarbeitende Validierungsregel eine Instanz von clsParser und ruft deren Parse()-Methode auf.

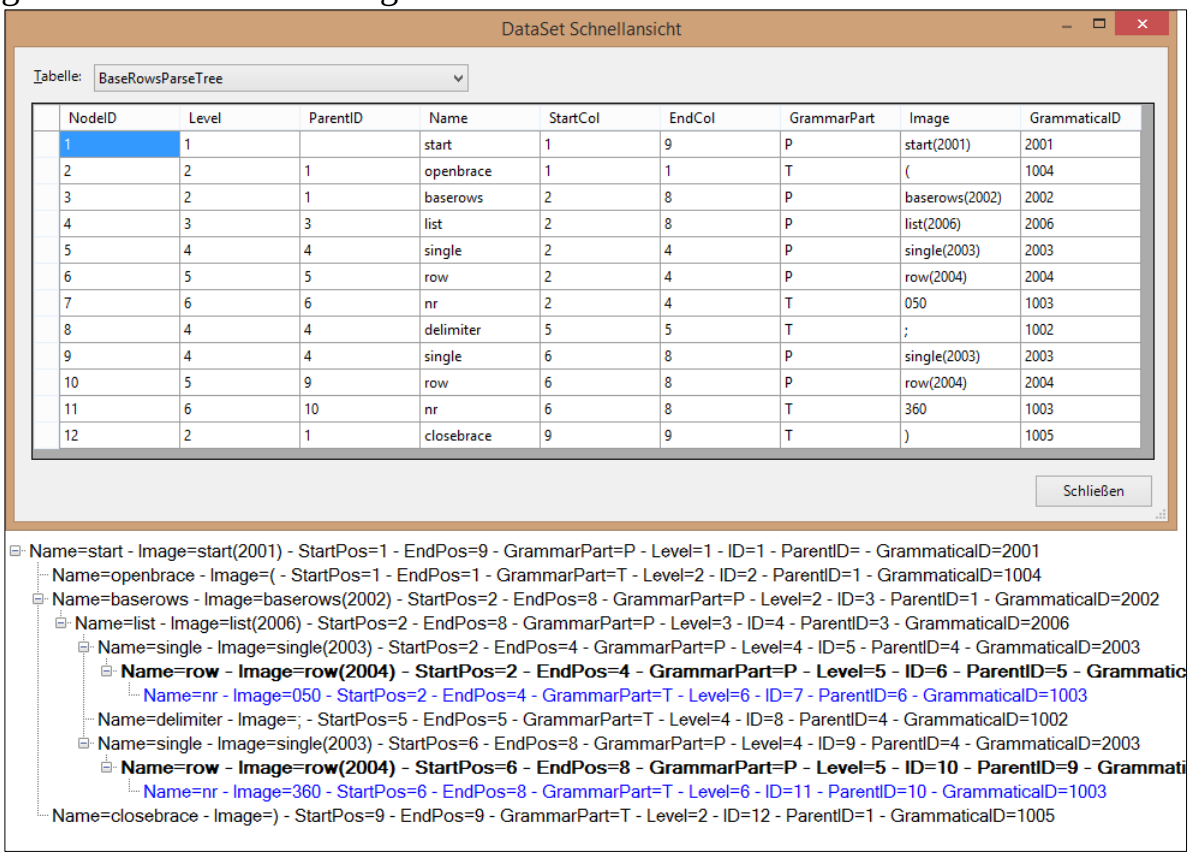
Abbildung 19 stellt diese beiden Klassen als UML-Klassendiagramme vor. Die Instanz von clsParserController erhält vom Aufrufer über Properties u. a. die zu parsenden Regeln und die ID der zu nutzenden DPM-Zellmenge. Die 2849 clsParser-Instanzen zeigen je ein DataSet als lokale Arbeitsdatenbank, ein Objekt für die zu untersuchende EBA-Validierungsregel, das DPM-Zellverzeichnis und 4 Parser für die Grammatiken. In den Methodennamen klingen die Arbeitsschritte an.

Abbildung 19: UML-Klassendiagramme der Klassen zu Parsersteuerung

Quelle: erstellt mit Doxygen aus den beiden Klassen des Parceval-Codes

4.1 Erzeugen der Parsebäume und Ablage in DataTables

Jeder Instanz der clsParser wird über eine DataRow-Property die zu untersuchende Validierungsregel übergeben. Mittels der 4 Parserobjekte, die sie ebenfalls als Properties erhält, erzeugt sie aus den 4 Regeltexten 4 Parsebäume. Diese werden dann in 4 DataTables (BaseTablesParseTree, BaseRowsParseTree, BaseColumnsParseTree und FormulaParseTree) des klasseninternen DataSets abgelegt, um sie für LINQ zugänglich zu machen. Abbildung 20 zeigt den Aufbau der DataTables am Beispiel der DataTable BaseRowsParseTree sowie einen Parsebaum, wie ihn Parceval darstellt. Erkennbar ist, dass die Eigenschaften der Nodes im Parsebaum in Spalten der DataTable übersetzt wurden, wie sich die Baumstruktur in den Spalten NodeID und ParentID widerspiegelt und wie der daraus entstehende TreeView den Parsebaum optisch aufbereitet. Die Hervorhebungen in Fettdruck und Schriftfarbe unterscheiden die relevanten Angaben vom „noise“ der rein grammatischen Herleitungsschritte.



NodeID	Level	ParentID	Name	StartCol	EndCol	GrammarPart	Image	GrammaticalID
1	1		start	1	9	P	start(2001)	2001
2	2	1	openbrace	1	1	T	(	1004
3	2	1	baserows	2	8	P	baserows(2002)	2002
4	3	3	list	2	8	P	list(2006)	2006
5	4	4	single	2	4	P	single(2003)	2003
6	5	5	row	2	4	P	row(2004)	2004
7	6	6	nr	2	4	T	050	1003
8	4	4	delimiter	5	5	T	;	1002
9	4	4	single	6	8	P	single(2003)	2003
10	5	9	row	6	8	P	row(2004)	2004
11	6	10	nr	6	8	T	360	1003
12	2	1	closebrace	9	9	T	)	1005

Abbildung 20: DataTable und TreeView eines BaseRows-Parsebaums im Vergleich

Quelle: eigene Darstellung im Debugger-Überwachungsfenster und Formular frmParser von Parceval. Der Eingabetext ist (050;360).

Die übrigen ParseTree-DataTables sind identisch aufgebaut, da auch die Parsebäume strukturell immer gleich sind: es sind Bäume von Grammatica-Nodes, vgl. Kapitel 3.5 sowie Abbildung 21.

4.2 Basiskriterien

Bei den 3 Basiskriterien ist die Ermittlung der benötigten Suchkriterien aus den DataTables der Parsebäume vergleichsweise einfach:

- Alle Basisrestriktionen wirken dimensional (Tabelle, Zeile, Spalte).
- Relevant sind letztlich nur Produktionen für Einzelwert, Liste und Bereich.
- Die Verschachtelungstiefe der Parsebäume ist durch die Nichtrekursivität der Grammatiken begrenzt.
- Ein Einzelwert lässt sich als Spezialfall eines Bereichs mit identischen Grenzen auffassen: 050 entspricht einem Bereich 050-050.
- Eine Basisrestriktion besteht somit immer aus einer **Bereichsliste** mit mindestens einem Eintrag oder dem Sonderfall „(All)“.

Damit genügt es, für die 3 Basiskriterien jeweils eine flache (nicht-reflexive) DataTable im DataSet dstParser aufzubauen. Die DataTables Base-Restrictions_Rows und BaseRestrictions_Columns erhalten zwei Spalten für die Bereichsgrenzen. Die DataTable BaseRestrictions_Table besteht sogar nur aus einer Spalte für den Tabellennamen, da es keine Tabellenbereiche gibt. Die Gesamtschau der bisher eingeführten DataTables zeigt Abbildung 21.

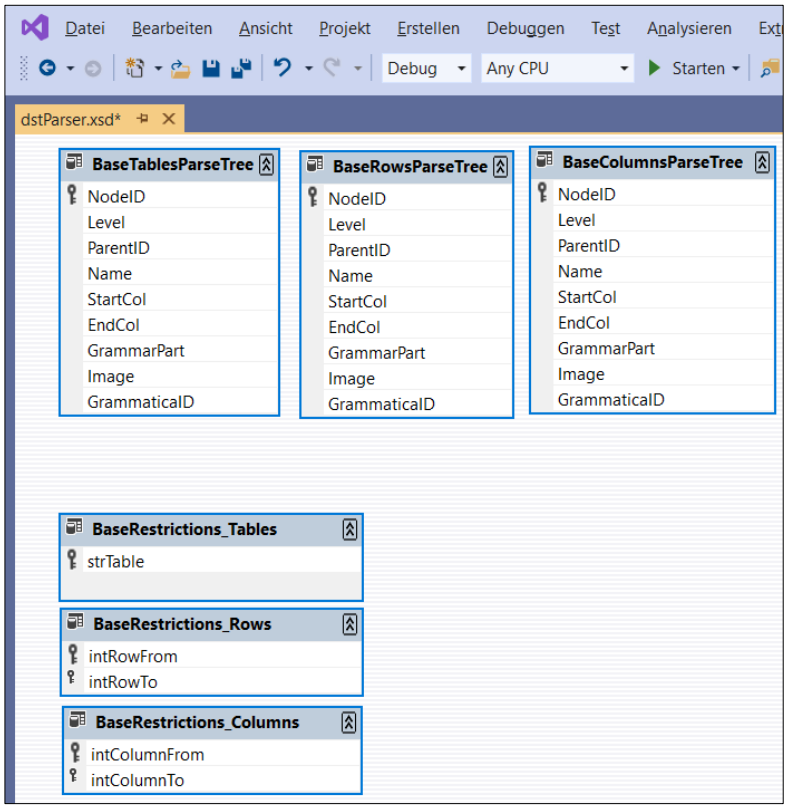


Abbildung 21: DataSet dstParser (Auszug)

Quelle: Bildschirmfoto von Parcevals DataSet dstParser in MS Visual Studio

Quelle und Ziel der Basiskriterienverarbeitung stehen damit fest: von den Parent-Child-DataTables (oben in Abbildung 21) in die DataTables der Suchkriterien (unten in gleicher Abbildung). Wie aber vorgehen?

Die Vorgehensweise beschreibt Parr als „*tree walk*“ [Parr_LIP, S. 7]. Der Parsebaum wird gleichsam abgeschritten, und bei jedem relevanten Knoten wird eine Reaktion ausgelöst. Das Prinzip eines **tree walks** wird in Abbildung 22 deutlich. Der gezeigte Parsebaum der Basiszeilenrestriktion (030;040) wird entlang des Pfeils durchschritten. Nur die Produktion *single* ist hier relevant; ihr Erreichen löst das Anfügen an die flache Liste der Basiszeilenkriterien aus (= Sternsymbol). Ist der Kindknoten der auslösenden *single*-Produktion ein *rowrange*-Knoten, werden aus dessen *row*-Kindknoten die Grenzen gelesen und als Bereich (von-bis) an die Liste angefügt. Hat der *single* dagegen eine *row* als Kind, entsteht ein Bereich mit gleichen Von- und Bis-Werten. Andere Nachkommen als *row* oder *rowrange* sieht die Basiszeilen-Grammatik für *single*-Knoten nicht vor, ebenso kommen keine rekursiven Abstiege vor[6].

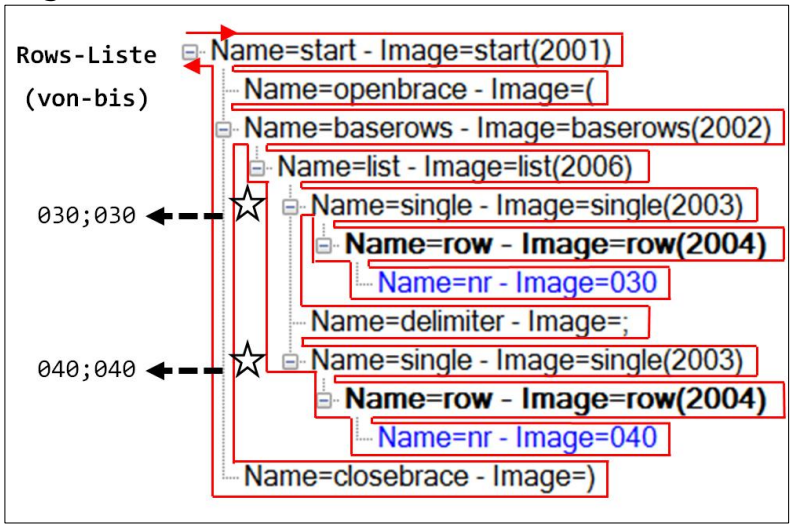


Abbildung 22: tree walk des Parsebaums der Basiszeilenrestriktion (030;040)

Quelle: eigenes Werk aus einem Parceval-Bildschirmfoto, angelehnt an [Parr_LIP, S. 103].

Das Ergebnis des tree walk, die erkannten Basisrestriktionen, verdeutlicht Parceval auch dem Programmanwender, wie Abbildung 23 zeigt. Die zugehörigen Parsebäume finden sich auf den ersten drei Registerkarten, die gezeigte vierte Registerkarte zeigt das Ergebnis. Die Tabellenraster zeigen die Inhalte der 3 BaseRestrictions-DataTables sowie die Zellen des DPM, die diese Kriterien erfüllen: die 14 Basiszellen.

---

6 Aus diesem Grund lässt sich diese Grammatik – wie auch die anderen beiden Grammatiken der Basisrestriktionen – einfacher als reguläre Grammatik darstellen. Die Analyse könnte dann mit regulären Ausdrücken geschehen.

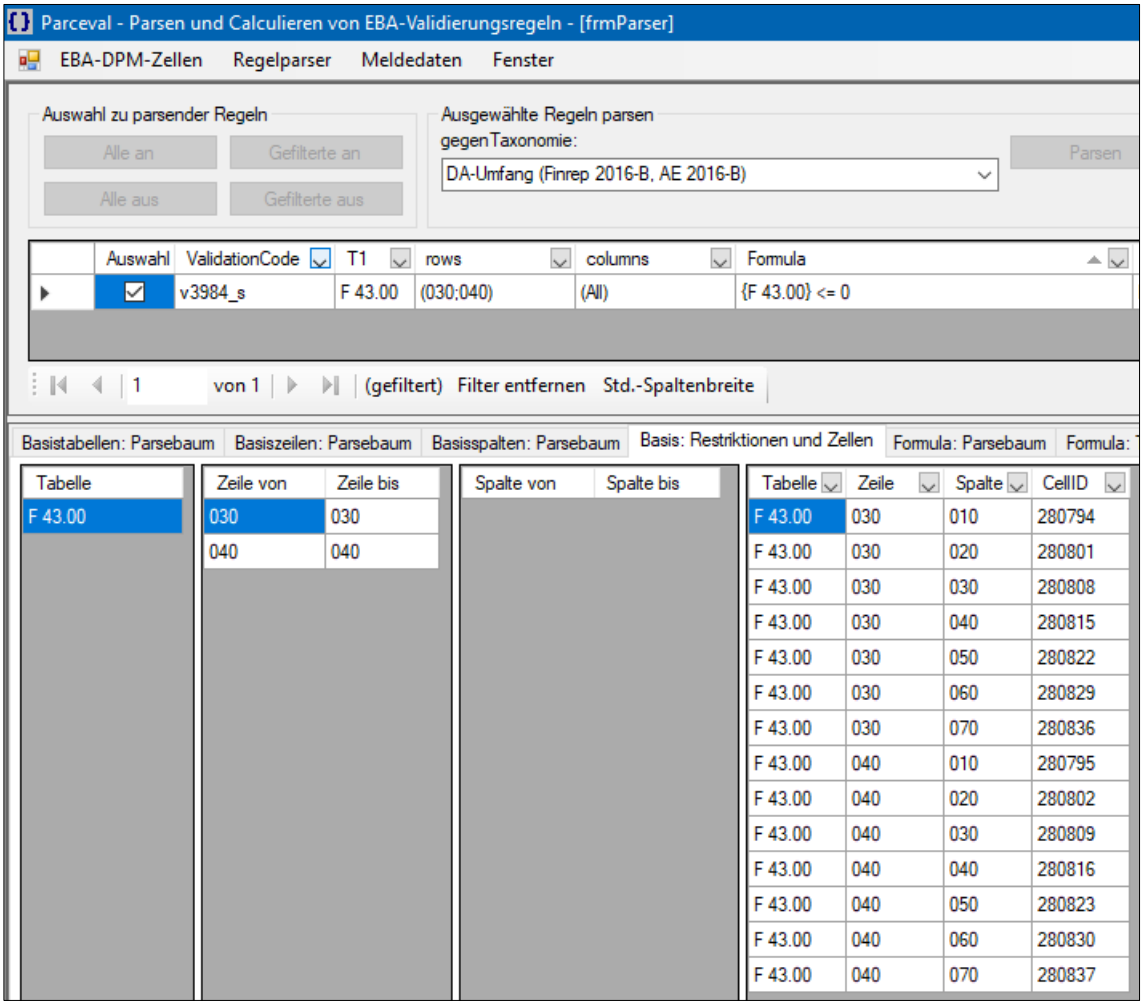


Abbildung 23: Erkannte Basiskriterien und resultierende Basiszellen

Quelle: Bildschirmfoto aus Parceval beim Parsen der EBA-Regel v3984_s

Die BaseRestrictions-DataTables werden nun auf den gesamten EBA-Zellvorrat angewendet, indem sie sukzessiv in einen LINQ-Join eingehen. Jede Basisrestriktion soll die bisher ermittelte Zellmenge weiter eingrenzen, allerdings ist dabei zu beachten, dass die BaseRestrictions-DataTables auch leer sein können. Dies geschieht genau dann, wenn das betroffene Basiskriterium leer oder nicht eingrenzend wirksam ist[7]. Solche leeren Kriterien würden im Join zu einem leeren Ergebnis führen. Da eine leere Restriktion im Falle der Validierungsregeln jedoch für *keine Einschränkung* (nicht für *keine Ergebnisse*) steht, dürfen die DataTables dieser Begrenzungen nur join-wirksam werden, wenn sie nicht leer sind. Dies erfordert sequenzielles Vorgehen, ein Gesamtjoin wäre fehl am Platz.

[7] Der in Zeilen- und Basisrestriktionen vorkommende Wert „(All)“ ist ein Sonderfall. Er wirkt nicht eingrenzend, beeinflusst aber dennoch die weitere Verarbeitung.

Einen Einblick in die Umsetzung im C#-Code zeigt Abbildung 24. Der LINQ-Query qryBaseCells wird initial der gesamte DPM-Zellvorrat der EBA (rote Menge) zugewiesen. In der 1. Begrenzung wird sie dann auf die nichtleere Liste der Basistabellen reduziert. Das Ergebnis (grüne Menge) wird mit der nichtleeren Liste der Basiszeilen zur violetten Menge verbunden: Bei relevanten Zellen liegt die Zeilenkoordinate in den Bereichsgrenzen. Analog geschieht dies schließlich mit den Basisspalten (blaue Menge, Code dazu nicht im Bild). Am Ende enthält qryBaseCells die Menge der Basiszellen – die Zellen, auf die die Formel angewendet werden muss.

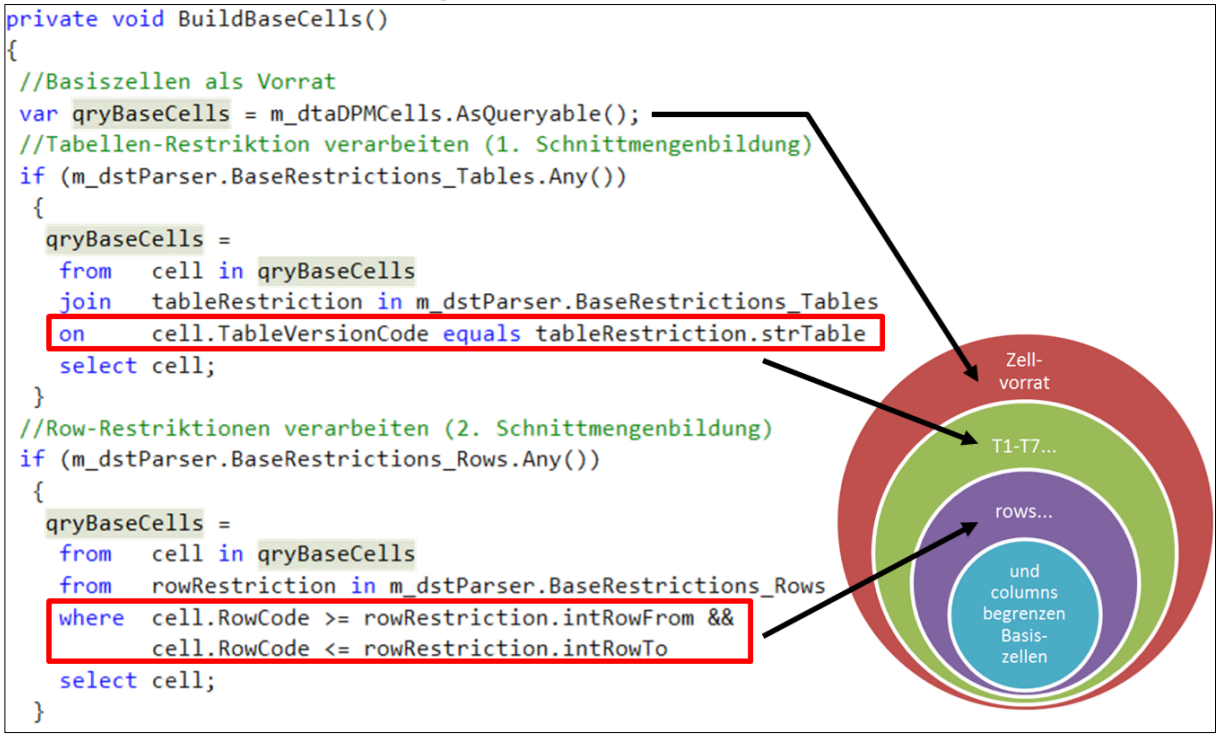


Abbildung 24: Sequenzielle Schnittmenge der Basiskriterien (Auszug)

Quelle: Eigenes Werk unter Nutzung eines Codefragments aus Parceval

4.3 Formel

Für die Formel einer EBA-Validierungsregel ist die Extraktion der relevanten Informationen aus ihrem Parsebaum komplexer als bei den Basiskriterien. Gründe dafür sind:

- Die Formeln entstehen aus einer **kontextfreien Grammatik**. Die Verschachtelungstiefe ihrer Parsebäume ist daher nicht begrenzt wie bei den regulär darstellbaren Basiskriterien[8].

[8] Die Basiskriterien sind zwar als kontextfreie Grammatiken implementiert worden, wären aber auch schon durch reguläre Grammatiken darstellbar gewesen.

- Die Parsebäume der Formeln enthalten viele Einträge, die zwar **grammatisch nötig, aber ohne Bedeutung** für die weitere Verarbeitung sind, z. B. Klammern, Kommas und andere Ordnungsstrukturen.

Auch hier gilt es, die relevanten Informationen vom grammatischen „noise“ ([Parr_LIP, S. 88]) zu trennen. Was also ist die Essenz einer Formel?

Die Lösung sind **abstrakte Syntaxbäume** (abstract syntax trees, **AST**), wie sie Parr in seinem Werk „Language Implementation Patterns“ beschreibt:
„The key idea behind AST structure is that tokens representing operators or operations become subtree roots. All other tokens become operands (children of operator nodes).“ [Parr_LIP, S. 76]

Ein abstrakter Syntaxbaum enthält somit nur noch das **Skelett der Struktur**:

- Operatoren – sie beschreiben, *was* zu tun ist und zeigen sich im AST als innere Knoten;
  und
- Operanden – sie beschreiben, *womit* dies zu tun ist und zeigen sich im AST als Nachfolgeknoten der Operatoren.

Wie schlank und übersichtlich ein AST sein kann, zeigt Abbildung 25. Auf den ersten Blick wird der **Vergleichsoperator als AST-Wurzel** deutlich, der auf beiden Seiten je genau einen **Term** als Operanden verarbeitet. Im linken Term ergibt sich durch Eingrenzung der Basiszellen auf die Zeilengrenze r010 genau eine Zelle; rechts ergeben sich analog 3 Zellen, die per Summe aggregiert werden. So entstehen die für einen arithmetischen Vergleich nötigen **Skalarwerte** auf beiden Seiten. Die insgesamt 4 ermittelten Zellbezüge werden als Platzhalter im AST gespeichert, ebenso die auszuführenden Berechnungen und Vergleiche. Das Validieren von Meldedaten geschieht später durch Einsetzen der Meldewerte in diese Platzhalter und anschließendes Ausführen geforderten Berechnungen.

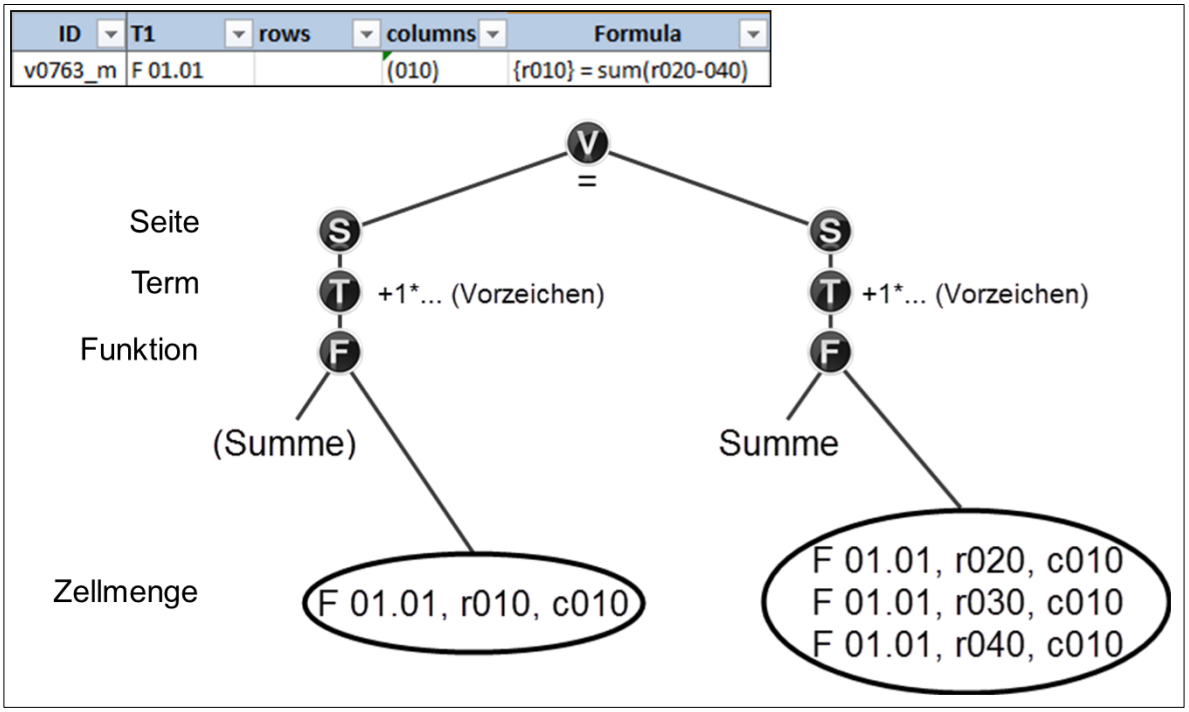


Abbildung 25: Abstrakter Syntaxbaum einer Regel

Quelle: eigenes Werk unter Nutzung von Regel v0763_m aus [EBA_VR_XLS]

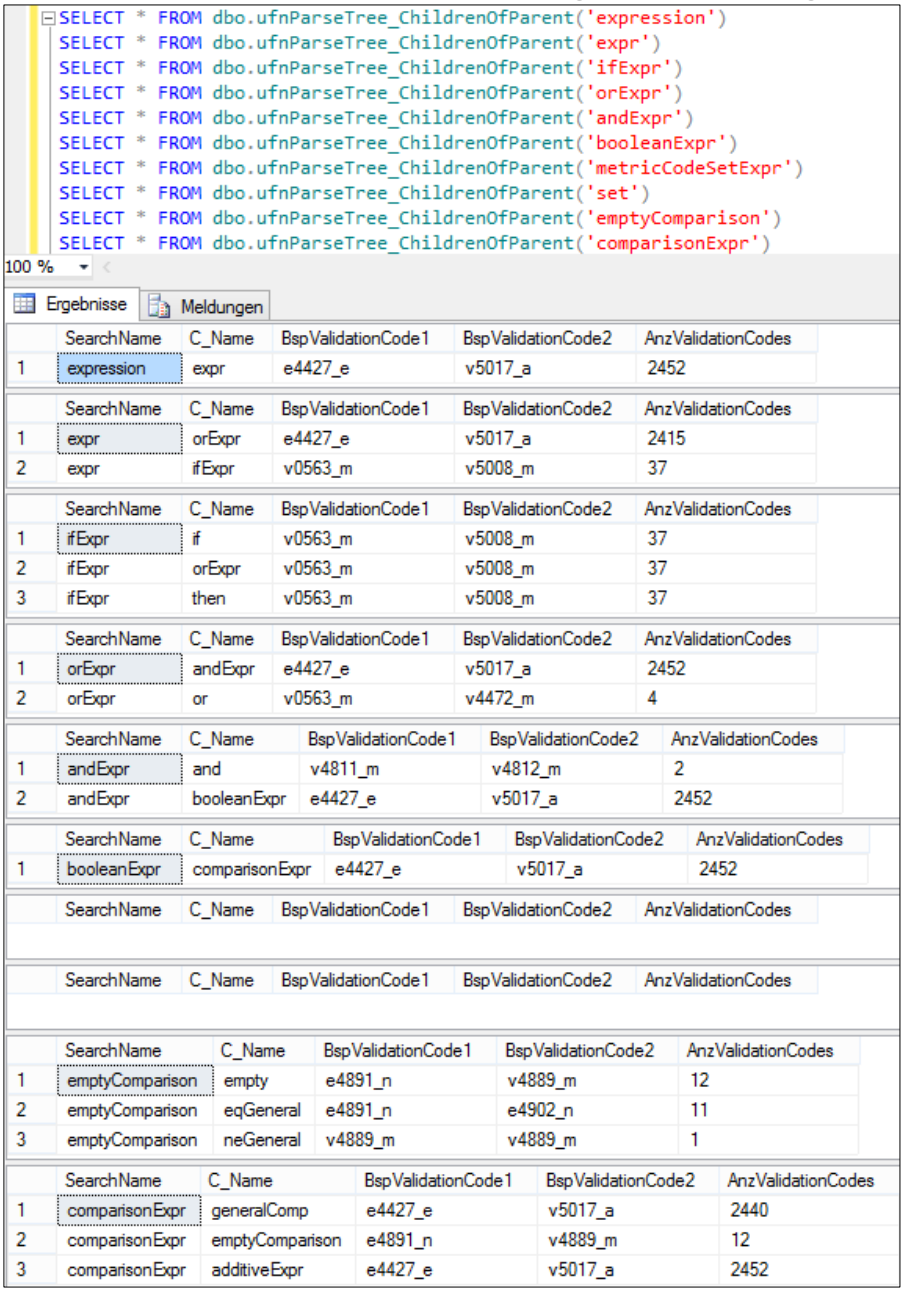
Die Untersuchung einiger Parsebäume ergab, welche Grammatikregeln AST-relevant und welche entbehrlicher „noise“ in Parrs Sinne sind:

- Operanden sind *Zellen* und *Literalwerte*. Nur sie evaluieren am Ende zu Werten, die verrechnet werden.
- Zu den Operatoren gehören *arithmetische* Anweisungen (+, -, *, /), Funktionen (sum, abs u. a.) sowie Vergleichsoperatoren (=, >, <= etc). Auch die *dimensionsbezogenen* Anweisungen wie row oder columnrange sind Operatoren, da sie Filterkriterien darstellen.
- Zum irrelevanten „noise“ gehören etwa die Tokens für Klammern und Komma sowie zahlreiche rein grammatisch bedingte Zwischen-produktionen im Parsebaum. Im AST werden sie z. T. durch Anordnung und Hierarchie der Knoten abgebildet.

Nun gilt es, ein Verfahren zu finden, das aus den umfangreichen Parsebäumen das Regelskelett erzeugt. Der Verfasser hat dazu alle Regeln von den Inspector-Klassen verarbeiten und die entstehenden 2452 Parsebäume[9] in eine einzige Datenbanktabelle ausgeben lassen. Diese Tabelle wurde dann

[9] Von den 2849 importierten EBA-Regeln erweisen sich nur 2452 als grammatikgerecht. Die übrigen 397 Regeln sind grammatikwidrig und ergeben keinen Parsebaum.

statistisch ausgewertet: welche Grammatikregeln kommen als Kinder welcher Produktionen wie oft vor? Das Ergebnis illustriert Abbildung 26. Die Häufigkeiten lassen Schwerpunkte erkennen, so sind z. B. nur 37 Regeln mit einem if-Ausdruck versehen, und nur 2 verwenden eine and-Konstruktion. Aus dieser Statistik und der Formelgrammatik leitet der Verfasser ab, wie der AST für Parceval grundsätzlich aussehen muss. Die selten aufkommenden Produktionen und Tokens – etwa für Multiplikation und Division – lässt er zunächst außer Acht. Diese sollten Gegenstand künftiger Entwicklungen sein.



```
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('expression')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('expr')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('ifExpr')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('orExpr')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('andExpr')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('booleanExpr')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('metricCodeSetExpr')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('set')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('emptyComparison')
SELECT * FROM dbo.ufnParseTree_ChildrenOfParent('comparisonExpr')
```

100 % — Ergebnisse | Meldungen

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	expression	expr	e4427_e	v5017_a	2452

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	expr	orExpr	e4427_e	v5017_a	2415
2	expr	ifExpr	v0563_m	v5008_m	37

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	ifExpr	if	v0563_m	v5008_m	37
2	ifExpr	orExpr	v0563_m	v5008_m	37
3	ifExpr	then	v0563_m	v5008_m	37

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	orExpr	andExpr	e4427_e	v5017_a	2452
2	orExpr	or	v0563_m	v4472_m	4

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	andExpr	and	v4811_m	v4812_m	2
2	andExpr	booleanExpr	e4427_e	v5017_a	2452

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	booleanExpr	comparisonExpr	e4427_e	v5017_a	2452

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	emptyComparison	empty	e4891_n	v4889_m	12
2	emptyComparison	eqGeneral	e4891_n	e4902_n	11
3	emptyComparison	neGeneral	v4889_m	v4889_m	1

	SearchName	C_Name	BspValidationCode1	BspValidationCode2	AnzValidationCodes
1	comparisonExpr	generalComp	e4427_e	v5017_a	2440
2	comparisonExpr	emptyComparison	e4891_n	v4889_m	12
3	comparisonExpr	additiveExpr	e4427_e	v5017_a	2452

Abbildung 26: Häufigkeiten der Folgeknoten (Auszug)

Quelle: eigene Darstellung; Ausführung der gezeigten SQLs gegen die Parceval-Datenbank

Für den AST einer Formel erkennt der Autor einen Aufbau, der in Abbildung

27 verdeutlicht wird. Einstiegspunkt ist der Vergleichsoperator (Produktion **comparisonExpr** der EBA-Grammatik), der 3 Operanden benötigt: einen Additivausdruck (**additiveExpr**) als linke Seite, einen allgemeinen Vergleich (**generalComp**) und eine weitere additiveExpr als rechte Seite. Der generalComp kann nur zu einem der Vergleichstokens weiterentwickelt werden (=, > etc). Die additiveExpr kann sich auf beiden Vergleichsseiten in ein oder mehrere **multiplicativeExpr** weiterentwickeln, die ggf. durch Tokens für Addition (+) oder Subtraktion (-) verkettet sind. Eine multiplicativeExpr lässt sich also als **mathematischer Term** auffassen. Ein solcher Term mündet stets in einen unären Ausdruck (**unaryExpr**), der sich wiederum in ein optionales Vorzeichen mit folgendem Primärausdruck (**primaryExpr**) zerlegt. Diese primaryExpr lässt sich als **numerischer Skalarwert** auffassen. An dieser Stelle endet die arithmetische Verarbeitung im AST, und die dimensionale Wertermittlung aus den Basiszellen beginnt. Numerische Skalarwerte können als Aggregat einer durch Dimensionskriterien eingegrenzten Zellmenge entstehen. Sie entstehen aber auch durch Bezug auf eine einzelne Zelle oder Nennung einer Literalzahl, wie es bei EBA-Regeln zur Vorzeichenprüfung (Type „sign“) geschieht: Diese vergleichen Zellen mit der Literalzahl 0.

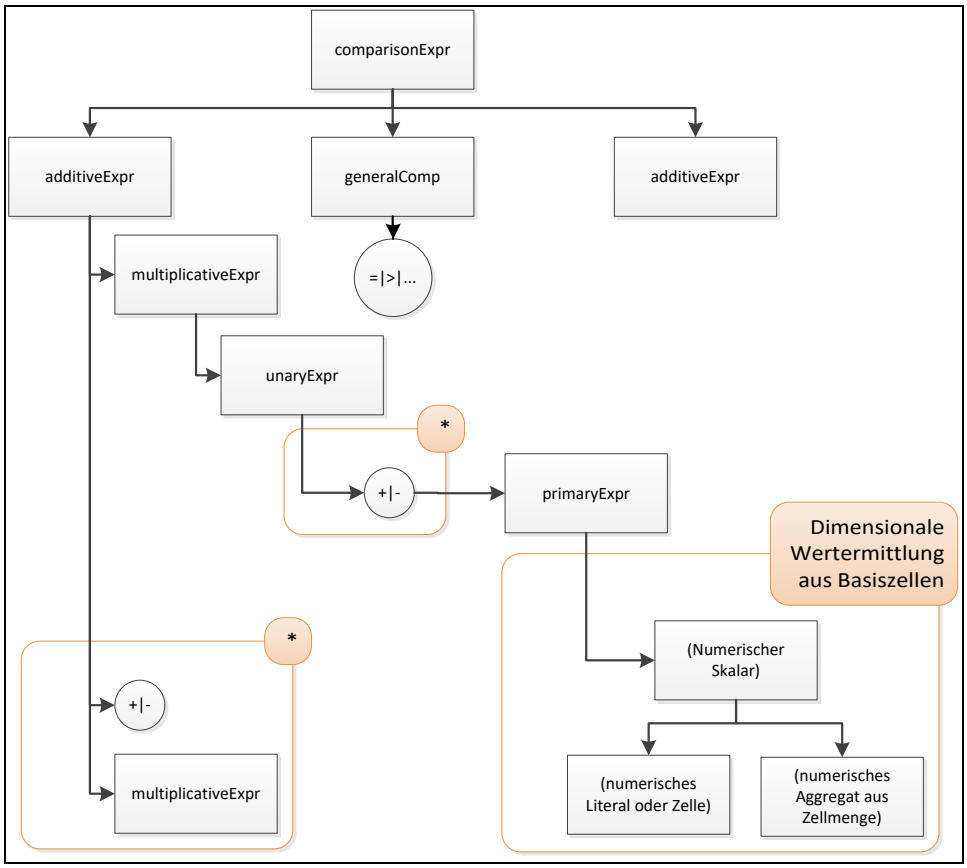


Abbildung 27: Aufbau des abstrakten Syntaxbaums der Formel (Regelskelett)

Quelle: eigene Darstellung unter Nutzung der Grammatikteile von [EBA_VR_Grammar]

Diese Schablone für den abstrakten Syntaxbaum wird von clsParser auf jede Regel angewendet. Für die dabei entstehenden Angaben (Vergleichsoperator, vorzeichenbehaftete Terme, Funktionen und Zellen) werden Datenbanktabellen benötigt, um sie zu persistieren. Dieses Teildatenmodell von Parceval wird nun kurz vorgestellt, denn das danach beschriebene Befüllungsverfahren wird leichter verständlich, wenn die Zielstruktur bekannt ist.

4.3.1 Teil-Datenmodell der ASTs

Abbildung 28 zeigt das Teildatenmodell von Parceval, in dem die ASTs abgelegt werden. Von links nach rechts gelesen werden die dargestellten Tabellen immer detaillierter, d. h. die 1-Seite einer Beziehung steht immer links. Das Modell beginnt bei Tabelle **RuleCompilations**, die Regelsammlungen bezeichnet. Bei jedem Speichern geparster Regeln in die Parceval-Datenbank entsteht eine neue RuleCompilation, die z. B. den Stand der verarbeiteten EBA-Regel-Exceldatei als Caption nennen könnte. Eine RuleCompilation hat mehrere **Rules** – dies ist somit die Tabelle, die Validierungsregeln aufnimmt. Ihre Attribute umfassen zu Dokumentationszwecken die ursprünglichen EBA-Vorgaben (ValidationCode, T1-T7, rows, columns, Formula u. a.), dazu den ComparisonOperator als erkannten Einstiegspunkt in den AST. Eine Regel besteht aus mindestens zwei Termen (**RulePartitions**). Jeder Term benennt seine Seite (Side aus L oder R), das Vorzeichen (Factor), seine Position in der Formel (TermRank, z. B. 1 für den 1. Term, über den Operator hinaus zählend) und bis zu zwei Funktionen (Aggregat- und/oder Skalarfunktionen), die er auszuführen hat. Diese Function1 und Function2 genannten Attribute erlauben ein Verschachteln von termbezogenen Funktionen: Function1 ist die äußere, Function2 die innere. Damit erlaubt Parceval z. B. einen Term, der einen Absolutwert einer Summe bildet: Function1 wird abs(), Function2 wird sum()[10]. Schließlich nimmt Tabelle **RulePartitionCells** die Literalwerte und DPM-Zellen auf, die der einzelne Term anspricht, sowie evtl. vorhandene zellbasierte Skalarfunktionen[11]. Literalwerte entstehen aus festen Werten ohne Zellbezug; so vergleichen etwa die zahlreichen Vorzeichenregeln mit dem Literalwert 0.

---

[10] Mehr als 2 Verschachtelungsebenen von Funktionen in einem Term werden folglich von Parceval nicht unterstützt, kamen im Untersuchungsumfang aber auch nicht vor.

[11] Beispiel: v1031_m, die in Abbildung 3 vorgestellt wurde, nutzt in einem Term eine Summe von Absolutwerten von Zellen: xsum(abs({…})). Hier ist die Summe termweit gültig (Function1), die Absolutfunktion jedoch zellbezogen (CellBasedScalarFunction), Function2 bleibt leer. Lautete der Term dagegen abs(xsum({…})), wären beide Funktionen termweit, so dass Function1 mit abs und Function2 mit sum belegt wäre und die CellBasedScalarFunction leer bliebe.

Sind jedoch Zellen angesprochen, verweisen die RulePartitionCells-Sätze auf Tabelle **DPM_Cells**, die alle Eigenschaften der DPM-Zellen nennt (Koordinaten, fachliche Bedeutung). Da das DPM von der EBA immer wieder weiterentwickelt wird, sind seine Zellen durch die Tabelle **DPM_TaxonomyVersions** versioniert. Die untersuchte Version 2.6 des DPM ist der derzeit einzige Eintrag in dieser Tabelle – künftige DPM-Versionen werden weitere erzeugen.

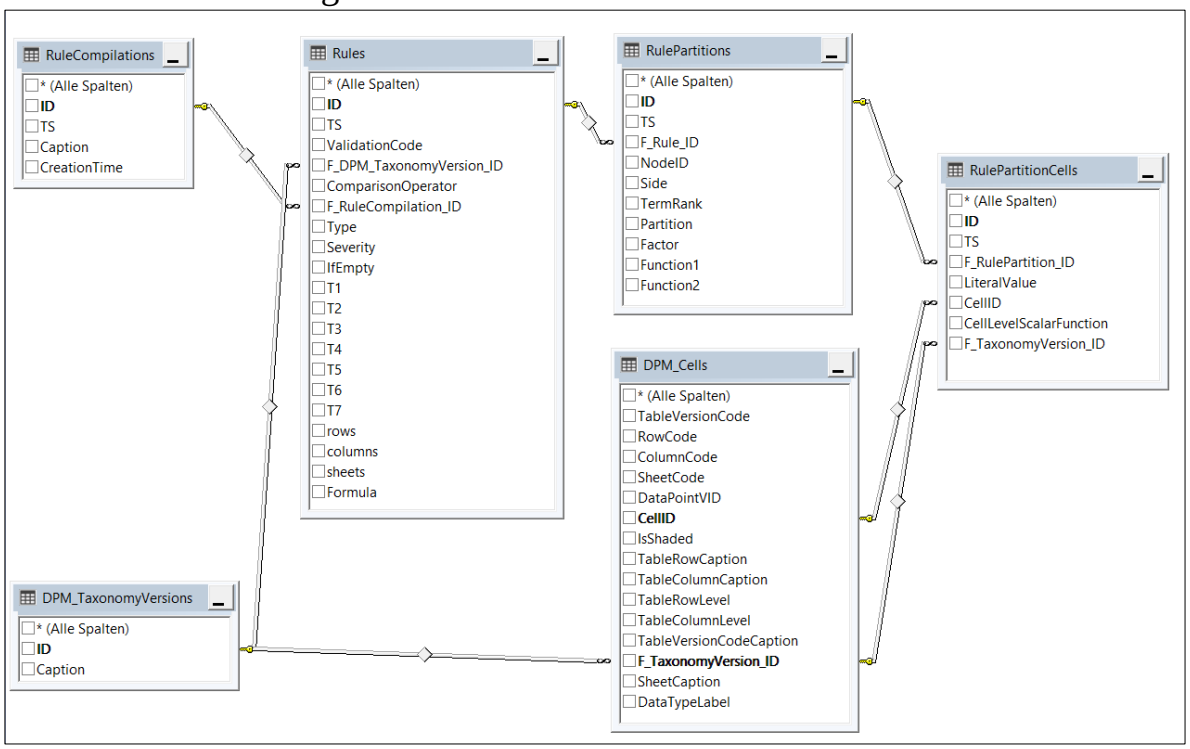


Abbildung 28: Teildatenmodell für die ASTs der EBA-Validierungsregeln

Quelle: eigenes Werk; Bildschirmfoto des Datenbankdiagramms von Parceval

Vergleicht man das Datenmodell in Abbildung 28 mit dem abstrakten Syntaxbaum in Abbildung 25, lassen sich Entsprechungen finden:

- Die *RulePartitionCells* entsprechen im AST den Zellmengen, die eine Regel insgesamt untersuchen muss.
- Diese Zellen sind im AST Termen zugeordnet – im Datenmodell sind die *RulePartitions* das Äquivalent zu Termen. Terme verlangen bei Bedarf auch Vorzeichen und Funktionen, wofür die Tabelle RulePartitions entsprechende Attribute vorsieht.
- Wurzel des AST einer Regel ist stets der Vergleichsoperator, den die Tabelle *Rules* aufnimmt.

4.3.2 Befüllung des AST-Datenmodells

Zwei wesentliche Aufgaben müssen beim Befüllen der AST-Tabellen aus den DataTables der Parsebäume gelöst werden:

- Die **Relevanz jeder Grammatikregel** für den AST muss bekannt sein. Grammatikteile wie Klammern oder Kommas sind irrelevant für den AST, arithmetische und dimensionale Grammatikteile jedoch bedeutsam.
- Die DataTable des Formel-Parsebaums muss **hierarchisch auswertbar** sein. Wird etwa wie in Abbildung 29 in einem Term eine Summe (Produktion sumFunction, Knoten-ID 23 im Bild) erkannt, finden sich die dimensionalen Suchkriterien in den Kindern und Kindeskindern dieses Parsebaumknotens. Parceval muss imstande sein, hierarchische Konstrukte wie Kinder, Nachkommen oder Elternknoten in der DataTable des Formel-Parsebaums zu ermitteln.

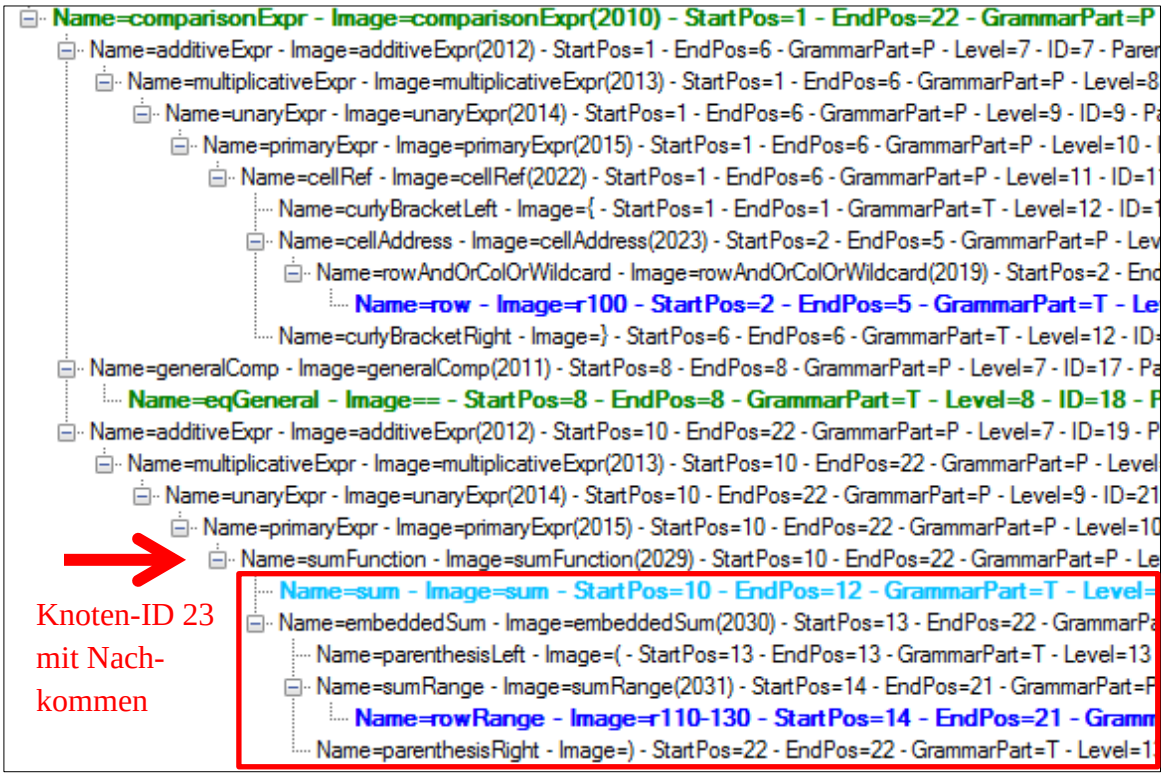


Abbildung 29: Hierarchische Suche im Formel-Parsebaum (Auszug)

Quelle: eigenes Werk; Parceval-Bildschirmfoto des Formel-Parsebaums von EBA-Regel v0765_m aus [EBA_VR_XLS]; rote Hervorhebung vom Verfasser. Der Formeltext lautet {r100} = sum(r110-130).

Für die AST-Relevanz aller Grammatikregeln legt der Verfasser die Datenbanktabelle **GrammarSymbols** an, erfasst hier alle 55 Tokens und 46 Produktionen und legt mit Bitfeldern deren Relevanz fest, vgl. Abbildung 30:

- table, row, rowrange, column, columnrange sind *dimensionsbezogene Filterkriterien* des Terms;
- Literale sind in der Formel *vorgegebene feste Werte*, sie wirken wie eine Einzelzelle (z. B. die 0 bei sign-Regeln: „… > 0“);
- sum und count sind *Aggregatfunktionen*;
- abs ist als *Skalarfunktion* anzusehen;
- plus (+) und minus (-) gelten als *Vorzeichen*;
- eqGeneral (=), gtGeneral (>) etc. sind *Vergleichsoperatoren*.

Aus Platzgründen nicht im Bild, aber ebenso in der Tabelle abgelegt:

- Klammern, Kommas u. a. sind *irrelevant* für den AST (alle Bits 0);
- set, if, varRef, mult, div u. a. werden von Parceval *derzeit nicht unterstützt* (Bitfeld IsNotSupportedByParceval = 1).

```
1  SELECT * FROM dbo.GrammarSymbols
2  WHERE GrammarName = 'ValidationFormula 2.6'
3  ORDER BY UsesDimensions DESC,   IsAggregateFunction DESC,  IsScalarFunction DESC,
4           IsNodeOperator DESC,   IsComparisonOperator DESC, IsNotSupportedByParceval DESC
```

100 %

Ergebnisse | Meldungen

	SymbolType	GrammaticalID	Image	UsesDimensions	IsAggregateFunction	IsScalarFunction	IsNodeOperator	IsComparisonOperator
1	T	1044	column	1	0	0	0	0
2	T	1045	colRange	1	0	0	0	0
3	P	2018	numericLiteral	1	0	0	0	0
4	T	1041	row	1	0	0	0	0
5	T	1042	rowRange	1	0	0	0	0
6	T	1051	table	1	0	0	0	0
7	T	1032	max	0	1	0	0	0
8	T	1033	min	0	1	0	0	0
9	T	1036	count	0	1	0	0	0
10	T	1035	sum	0	1	0	0	0
11	T	1039	xsum	0	1	0	0	0
12	T	1034	abs	0	0	1	0	0
13	T	1017	minus	0	0	0	1	0
14	T	1016	plus	0	0	0	1	0
15	T	1022	ltGeneral	0	0	0	0	1
16	T	1021	neGeneral	0	0	0	0	1
17	T	1023	leGeneral	0	0	0	0	1
18	T	1024	gtGeneral	0	0	0	0	1
19	T	1025	geGeneral	0	0	0	0	1
20	T	1020	eqGeneral	0	0	0	0	1
21	T	1019	div	0	0	0	0	0
22	T	1009	if	0	0	0	0	0

Abbildung 30: Datenbanktabelle GrammarSymbols (Auszug)

Quelle: Eigenes Werk; Bildschirmfoto der gezeigten SQL in SSMS 2012 gegen Parceval-Datenbank

Damit ist Parceval in der Lage, die AST-relevanten Teile von den irrelevanten zu unterscheiden.

Für die hierarchische Suche in den Parsebäumen erstellt der Verfasser einige Erweiterungsmethoden für LINQ und bündelt sie in der Klasse clsLINQExtensions. Wird die DataRow eines Parsebaumknotens übergeben, liefert beispielsweise

- **getChildren(ParseTreeRow)** deren Kinder,
- **getDescendants(ParseTreeRow)** deren gesamten Unterbaum (Nachkommen), d. h. die Kinder, Kindeskinder etc.,
- **getSiblings(ParseTreeRow)** deren Geschwister, d. h. Knoten mit demselben Parent wie der übergebene.

Diese Methoden haben Vorbilder in der mehrdimensionalen Abfragesprache MDX[12], im C#-Sprachbestandteil LINQ To XML, in der Klasse FastTreeView[13] und in LINQ To Tree ([Eberhardt_LTT]); umgesetzt wurden sie vom Verfasser als z. T. rekursive Methoden mit LINQ To DataSet. Die Methoden lassen sich mit Standard-LINQ-Techniken verbinden. Hat z. B. getDescendants() einen kompletten Unterbaum isoliert, lässt er sich mit LINQ-Operatoren wie Where, GroupBy oder Count weiter untersuchen.

Parceval macht in clsParser von diesem Ansatz regen Gebrauch, um alle für den AST relevanten Angaben aus der DataTable des Formel-Parsebaums auszulesen. Beispielhaft lassen sich die dimensionsbezogenen Nachkommen des Knotens mit ID = 23 mit folgendem LINQ-Ausdruck in C# ermitteln, vgl. Abbildung 29:

```
var TermDimensions = clsLinqExtensions
 .getDescendants(m_dstParser.FormulaParseTree.FindByNodeID(23))
 .Where(ParseTreeRow => clsLinqExtensions
 .usesDimensions(ParseTreeRow, _dtaGrammarSymbols));
```

Deutung dieser Codezeile: die DataRow, die den Parsebaumknoten mit ID 23 darstellt (Einstiegspunkt), wird als Argument an getDescendants() übergeben. Die gefundene Menge der Nachkommen (6 Knoten) wird mit einer Where-

---

12 MDX = Multidimensional Expressions, Abfragesprache für OLAP-Datenbanken in Microsoft SQL Server Analysis Services. Dort existieren z. B. die Funktionen Descendants, Siblings, Children und Ascendants. Die Analogie zu Familienstammbäumen mit Nachkommen, Geschwistern, Kindern und Vorfahren ist kein Zufall, sondern verdeutlicht, was die Funktionen in hierarchischen Dimensionen leisten.

13 Jakob Lithners Code zum FastTreeView, [Lithner_FTV], ist parceval-weit als schnelle TreeView-Variante im Einsatz, enthält aber auch eine quelloffene Methode getDescendants(), die die Nachfahren eines TreeView-Knotens ermittelt. Der Verfasser hat Lithners Algorithmus auf DataTables angewendet und den Namen übernommen.

Klausel begrenzt auf diejenigen Knoten, die in der Grammatiksymboltabelle als dimensionsbezogen markiert sind. Gefunden wird nur die rowrange.

Ähnlich gestaltet sich die Suche nach allen anderen Bestandteilen, die für den AST relevant sind: Nachkommen des Einstiegspunkts abrufen und nach AST-relevanten Kennzeichen filtern. In clsLINQExtensions entwickelt der Verfasser auch für deren Bit-Attribute geeignete Prädikatmethoden wie IsAggregateFunction() oder IsComparisonOperator().

Was aber sind geeignete **Einstiegspunkte**, deren Nachkommen zu untersuchen wären? Ein Blick zurück auf Abbildung 27 zeigt: Es genügt, den Vergleichsoperator zu erkennen und in dessen zwei Seiten jeden Term auf die relevanten Nachkommen zu durchsuchen.[14] Als Term gilt dabei die Produktion multiplicativeExpr.

Schon im Formeltext ist das anschaulich, etwa bei Regel v2927_m aus [EBA_VR_XLS]:

{F 36.01.b, r160, c180} + {F 36.02.b, r160, c190} =
{F 32.04.a, r090, c010} - sum({F 32.04.a, c010, (r100, r110)})

Diese Formel besteht aus einem Gleichheitsoperator, der links wie rechts zwei Terme enthält. Von den 4 Termen sind die ersten 3 offensichtliche Einzelzellbezüge, der letzte ist eine Summe von zwei Zellen einer Tabellenspalte.[15] Das erkennt auch Parceval durch obigen Ansatz. Die Ergebnisse werden in die Datenbanktabellen ausgegeben, und obige Regel zeigt sich auf einen Blick in Abbildung 31. Die obere Abfrage zeigt für die Regel den Vergleichsoperator (=), die aus den Baumknoten-IDs[16] 8, 22, 38 und 52 erkannten Terme 1 bis 4, die Einordnung links oder rechts des Vergleichsoperators sowie die Vorzeichen und Funktionen der Terme. Die untere Abfrage zeigt für die erkannten 5 Zellen jeweils den Term, in den sie eingehen. Der Summenterm nutzt zwei Zellen, die anderen Terme nur je eine.

---

[14] Dieser Ansatz hat Grenzen. Er setzt voraus, dass die Terme nur addiert oder subtrahiert werden. Für andere Operationen, etwas Multiplizieren oder Dividieren, würde das Durchsuchen der Nachkommen den Operatorvorrang ignorieren.

[15] Die hier weggelassenen Basiskriterien sind für das Nachvollziehen dieser Regel unnötig, da die Terme jeweils alle 3 Dimensionen eingrenzen.

[16] Die Baumknoten und ihre IDs sind in Abbildung 32 erkennbar.

```
SELECT Rules.ValidationCode, Rules.ComparisonOperator,
       RulePartitions.NodeID, RulePartitions.Side,
       RulePartitions.TermRank, RulePartitions.Factor,
       RulePartitions.Function1, RulePartitions.Function2
FROM   RulePartitions INNER JOIN Rules
       ON RulePartitions.F_Rule_ID = Rules.ID
WHERE  (Rules.F_RuleCompilation_ID = 99) AND
       (Rules.ValidationCode = 'v2927_m')
ORDER BY RulePartitions.NodeID

SELECT Rules.ValidationCode, RulePartitions.TermRank,
       DPM_Cells.TableVersionCode, DPM_Cells.RowCode, DPM_Cells.ColumnCode
FROM   RulePartitions
       INNER JOIN Rules
            ON RulePartitions.F_Rule_ID = Rules.ID
       INNER JOIN RulePartitionCells
            ON RulePartitions.ID = RulePartitionCells.F_RulePartition_ID
       INNER JOIN DPM_Cells
            ON RulePartitionCells.CellID = DPM_Cells.CellID AND
               RulePartitionCells.F_TaxonomyVersion_ID = DPM_Cells.F_TaxonomyVersion_ID
WHERE  (Rules.F_RuleCompilation_ID = 99) AND
       (Rules.ValidationCode = 'v2927_m')
ORDER BY RulePartitions.TermRank
```

100 %

Ergebnisse | Meldungen

	ValidationCode	ComparisonOperator	NodeID	Side	TermRank	Factor	Function1	Function2
1	v2927_m	=	8	L	1	1	NULL	NULL
2	v2927_m	=	22	L	2	1	NULL	NULL
3	v2927_m	=	38	R	3	1	NULL	NULL
4	v2927_m	=	52	R	4	-1	sum	NULL

	ValidationCode	TermRank	TableVersionCode	RowCode	ColumnCode
1	v2927_m	1	F 36.01.b	160	180
2	v2927_m	2	F 36.02.b	160	190
3	v2927_m	3	F 32.04.a	90	10
4	v2927_m	4	F 32.04.a	100	10
5	v2927_m	4	F 32.04.a	110	10

Die Abfrage wurde erfolgreich ausgeführt.

Abbildung 31: Regeldetails zu v2927_m in Parcevals Datenmodell

Quelle: Eigenes Werk; Bildschirmfoto der Abfrage aus Parceval-Datenbank

Zusammenfassend verdeutlicht Abbildung 32 eindrucksvoll, wie stark der abstrakte Syntaxbaum die Information des Parsebaums skelettiert. Die Abbildung zeigt den Parsebaum der soeben untersuchten Regel v2927_m – 77 Knoten lang – und die Essenz daraus für den AST. Den 4 Termen des Formeltextes entsprechen die blauen Zahlensymbole am Rand, die Operatoren zwischen den Termen sind rote Symbole am Rand, und die AST-Wurzel ist das schwarze Gleichheitszeichen am Rand. AST-relevante Äste im Parsebaum hebt Parceval farbig hervor; Irrelevantes bleibt im Parsebaum schwarz.

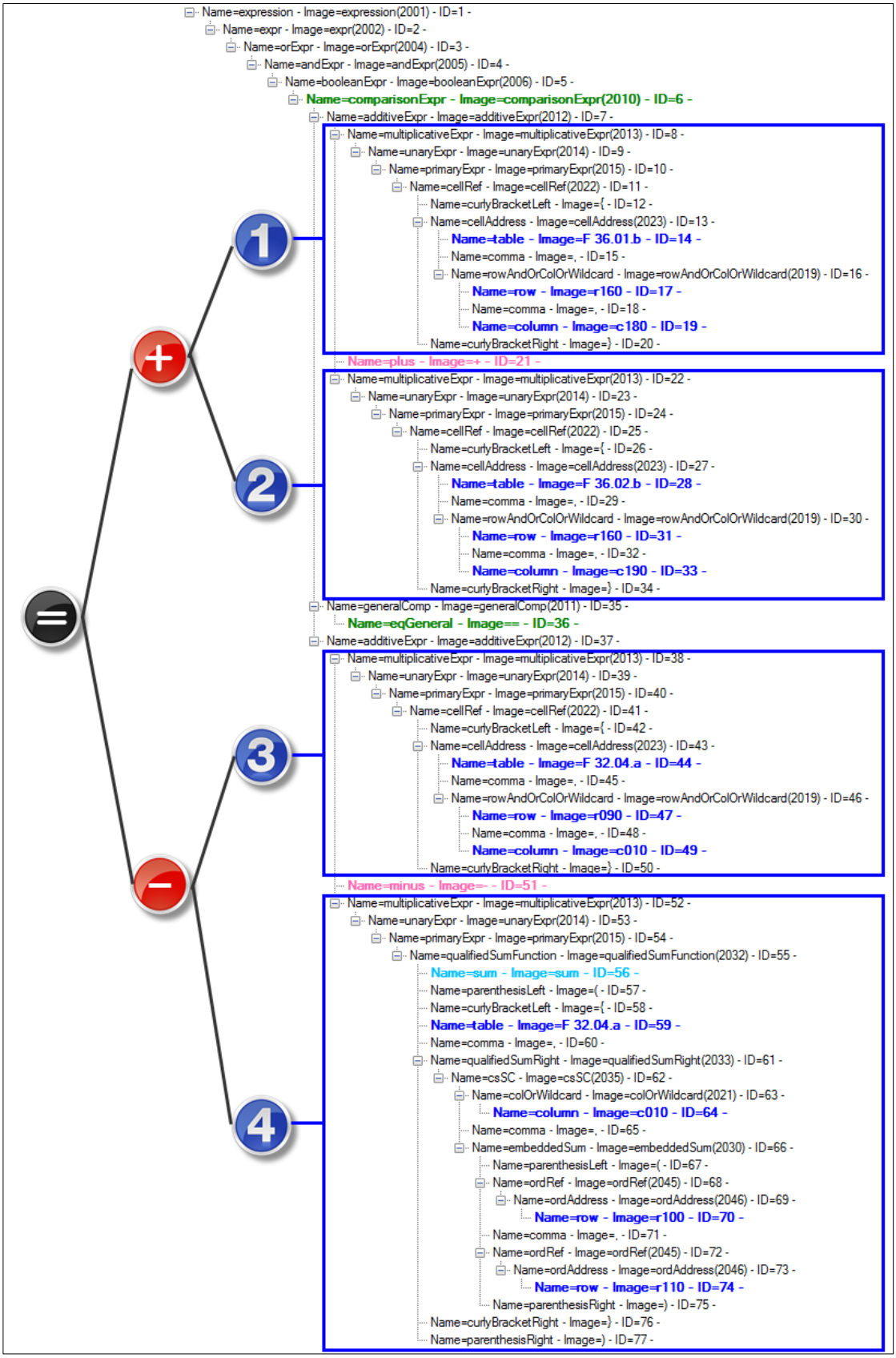


Abbildung 32: Zusammenhang zwischen Parsebaum und AST in Regel v2927_m

Quelle: Eigene Darstellung; grafisch ergänztes Bildschirmfoto des Formel-Parsebaums aus Parceval

Insgesamt untersucht Parceval alle Regeln auf Parsebarkeit und verarbeitet sie wie oben beschrieben. Der Vorgang dauert auf dem Rechner des Verfassers[17] für 2849[18] Regeln ca. 19 Sekunden.

Ein Bildschirmfoto der Ergebnisse zeigt Abbildung 33. Oben ist die Liste aller Regeln erkennbar, darunter die Details der in der Liste markierten Regel v2927_m. Links zeigt sich der erkannte Vergleichsoperator (=), darunter die 4 Terme mit Seiten (L/R) und Einstiegsknoten-ID im Parsebaum; der 4. Term ist markiert. Rechts davon zeigen sich zum markierten 4. Term oben das negative Vorzeichen (-1), darunter die Aggregatfunktion (sum), rechts oben die 3 Dimensionskriterien des markierten Terms, darunter die sich daraus ergebenden zwei Zellen als Teilmenge der Basiszellen. Leer, da im markierten Term ungenutzt, sind die Tabellen für Skalarfunktion und Literalwert.

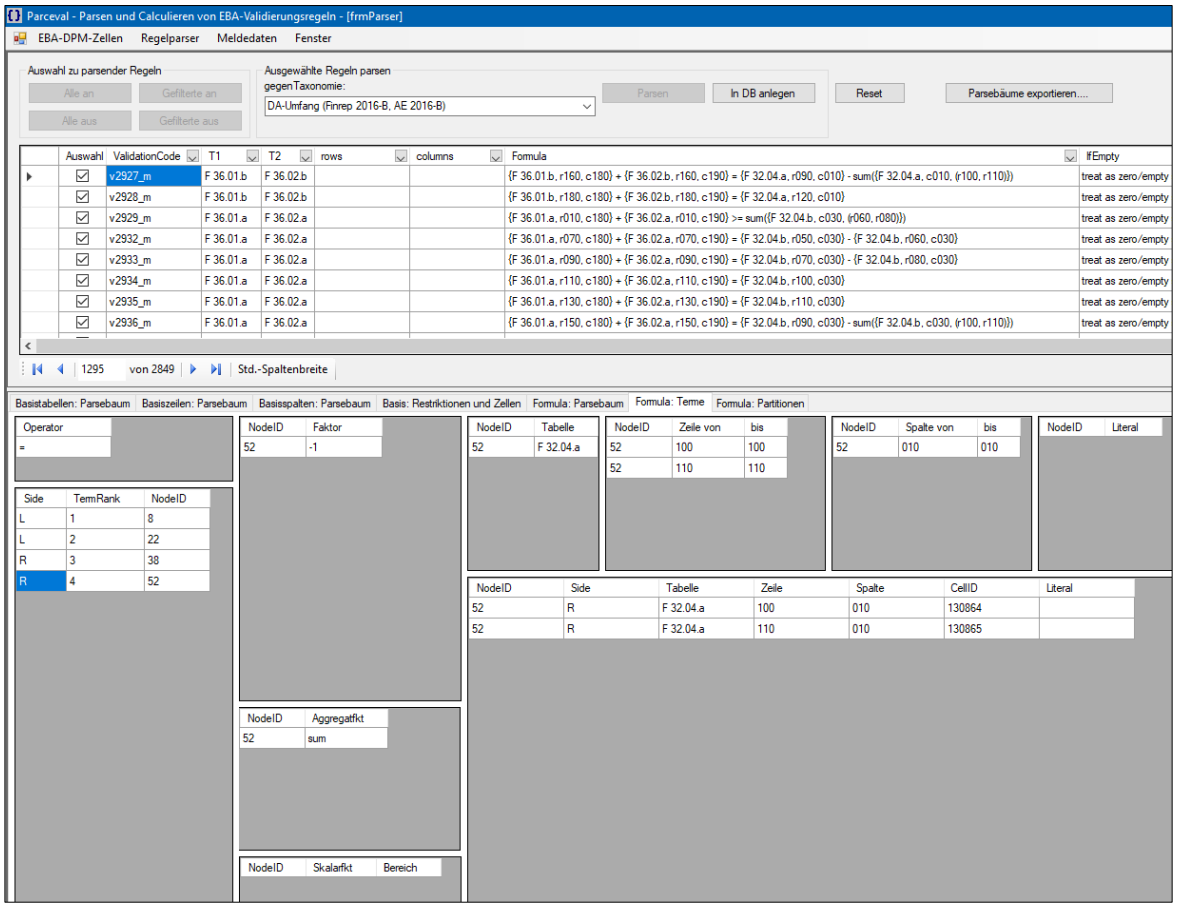


Abbildung 33: Parsen von Regeln in Parceval (Auszug)

Quelle: eigenes Werk, Bildschirmfoto von Parceval

---

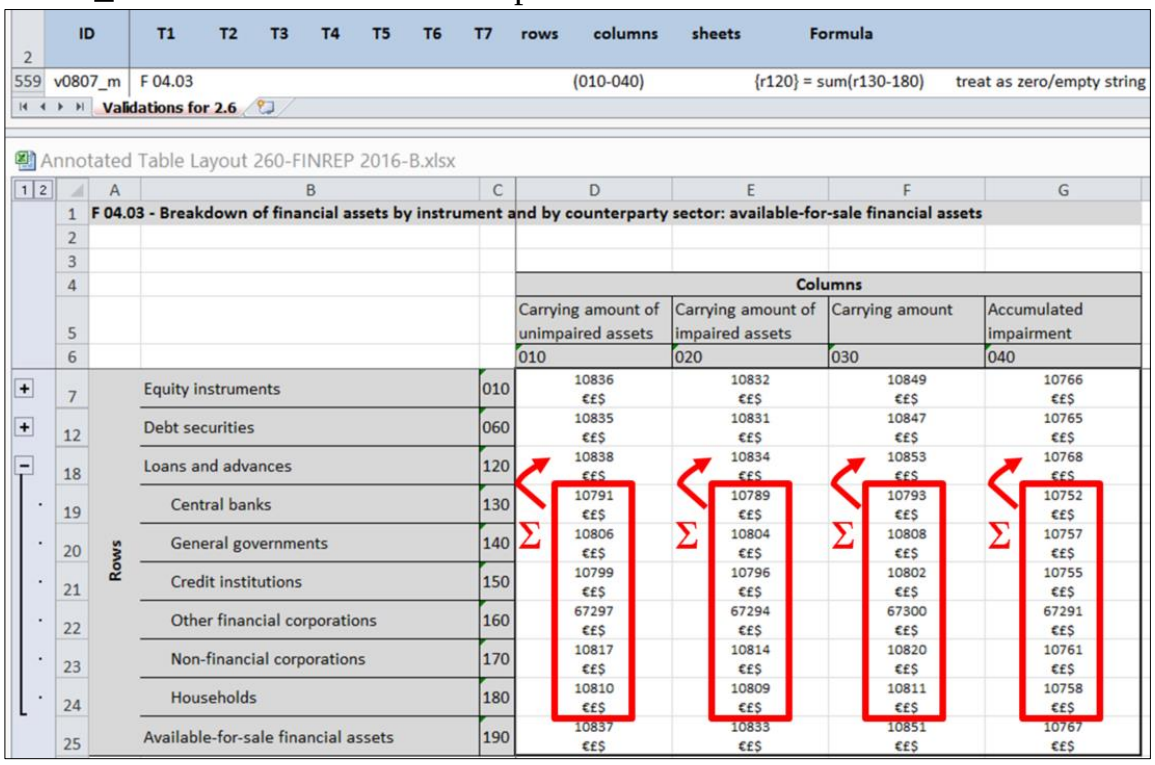
17 Notebook mit Intel-i7-7700HQ-Prozessor bei 2,8 GHz, 8 GB RAM, MS Windows 10 64-bit, MS VisualStudio 2012 Ultimate und MS SQL Server 2012 Enterprise Edition im lokalen Betrieb, Ausführung aus Visual Studio ohne Debugmodus.

18 inkl. der 397 Regeln, die sich beim Parsen als grammatisch falsch erweisen.

4.3.3 Partitionen (Exkurs)

Aus Platzgründen ist dieses Kapitel in diesem Dokument nur als Exkurs enthalten.

In 502 Regeln des Untersuchungsumfangs ergeben sich aus der Schnittmenge der Basiszellen mit den zusätzlichen Dimensionsgrenzen der Formelterme keine Skalarwerte für alle Terme. Für Nichtskalare ist ein arithmetischer Vergleich allerdings nicht anwendbar. Ein Beispiel ist Regel v0807_m, vgl. Abbildung 34. Auf der linken Seite des Vergleichs ergeben sich 4 Zellen, die mangels Aggregatfunktion keinen Skalarwert bilden. Hier soll jede der 4 Zellen einzeln mit ihrer zugehörigen Summe verglichen werden (vgl. rote Markierungen im Bild). Der Verfasser entwickelt hierfür ein Verfahren, das solche Regeln partitioniert. Parceval erkennt betroffene Regeln, findet den Partitionierungsvektor – im Beispiel die 4 Spalten der Tabelle – und teilt die gefundenen Terme auf in **Partitionen, die jeweils Skalare vergleichen**: hier also eine Zelle auf der linken Seite, eine Summe auf der rechten. Bei der späteren Validierung von Meldedaten ist eine partitionierte Regel nur dann erfüllt, wenn all ihre Partitionen erfüllt sind. Extremfall ist EBA-Regel v4358_s mit 396 Partitionen – sie prüft 396 Einzelzellen auf ihr Vorzeichen.



ID	T1	T2	T3	T4	T5	T6	T7	rows	columns	sheets	Formula	
v0807_m	F 04.03								(010-040)		{r120} = sum(r130-180)	treat as zero/empty string

Validations for 2.6

Annotated Table Layout 260-FINREP 2016-B.xlsx

F 04.03 - Breakdown of financial assets by instrument and by counterparty sector: available-for-sale financial assets

		Columns			
		Carrying amount of unimpaired assets	Carrying amount of impaired assets	Carrying amount	Accumulated impairment
		010	020	030	040
Equity instruments	010	10836 €£$	10832 €£$	10849 €£$	10766 €£$
Debt securities	060	10835 €£$	10831 €£$	10847 €£$	10765 €£$
Loans and advances	120	10838 €£$	10834 €£$	10853 €£$	10768 €£$
Central banks	130	10791 €£$	10789 €£$	10793 €£$	10752 €£$
General governments	140	10806 €£$	10804 €£$	10808 €£$	10757 €£$
Credit institutions	150	10799 €£$	10796 €£$	10802 €£$	10755 €£$
Other financial corporations	160	67297 €£$	67294 €£$	67300 €£$	67291 €£$
Non-financial corporations	170	10817 €£$	10814 €£$	10820 €£$	10761 €£$
Households	180	10810 €£$	10809 €£$	10811 €£$	10758 €£$
Available-for-sale financial assets	190	10837 €£$	10833 €£$	10851 €£$	10767 €£$

Rows

Abbildung 34: Partitionierung

Quelle: oben: [EBA_VR_XLS, Regel v0807_m], unten: [EBA_ATL_Finrep, Tabelle F 04.03], rote Markierungen sowie Ausblendung unnötiger Details durch den Verfasser

4.4 Anwenden auf Meldedaten (Validieren)

Parceval kann die in der Datenbank abgelegten Regeln auf Meldedaten anwenden, um zu prüfen, ob sie allen Regeln genügen (Validieren von Meldedaten). Erkennt Parceval in den Meldedaten Regelverstöße, ist eine Weitergabe der Daten an die Meldebehörde nicht sinnvoll, da diese die Einreichung voraussichtlich zurückweisen wird[19]. Eine Ursachensuche in den Quellsystemen und Zulieferprozessen der Bank ist dann angezeigt. Erkennt Parceval dagegen keine Fehler, können die Daten an die Aufsichtsbehörde geleitet werden.

Für Test- und Demonstrationszwecke wurde ein **Generator und Editor für Meldedaten** geschaffen. Durch Zufallszahlen werden fiktive Meldedaten erzeugt, die von Parceval validiert werden können. Diese Daten können ggf. manuell überarbeitet werden, etwa um Grenzfälle zu testen. In einer Produktionsumgebung würde dieser Meldedaten-Generator natürlich durch Zulieferungen aus Quellsystemen der Bank ersetzt; auch der Editor wäre im Produktivbetrieb nicht mehr akzeptabel, um die Meldungen prüfungs- und revisionssicher abzuwickeln.

Eine Meldung besteht prinzipiell aus der Zuweisung von Meldewerten zu DPM-Zellen, wie schon Abbildung 1 darlegte. Im Datenmodell von Parceval stellen sich Meldedaten in folgendem Teilmodell dar, vgl. Abbildung 35. Die bereits vorgestellten Tabellen **DPM_TaxonomyVersions** und **DPM_Cells** stellen den Wertevorrat aller Zellen des EBA-DPM dar. Die Meldewerte zu diesen Zellen finden sich in Tabelle **FilingData** als m:n-Auflösungstabelle zu Tabelle **Filings**. Ein Filing-Datensatz bündelt die Daten eines Meldestands, seine Caption könnte z. B. die Meldedaten des 1. Quartals von denen des Jahresendes unterscheiden oder auch mehrere Meldeläufe desselben Abschlusstages. Die Tabelle DPM_Cells ist den Teildatenmodellen von Meldedaten und Regelskeletten gemeinsam: Regeln prüfen Zellen, die von Meldungen mit Werten belegt werden. Das macht sich Parceval beim Validieren eines Filings zunutze.

---

[19] Da die EBA als Normsetzerin jedoch jederzeit den Regelbestand abändern kann, ist es denkbar, dass sie bei Einreichung eine neuere Version der Regeln anwendet und so zu einem anderen Ergebnis kommt als Parceval. Zeitnahe Importe aktueller Regeln sind für die Aussagekraft von Parceval daher wichtig.

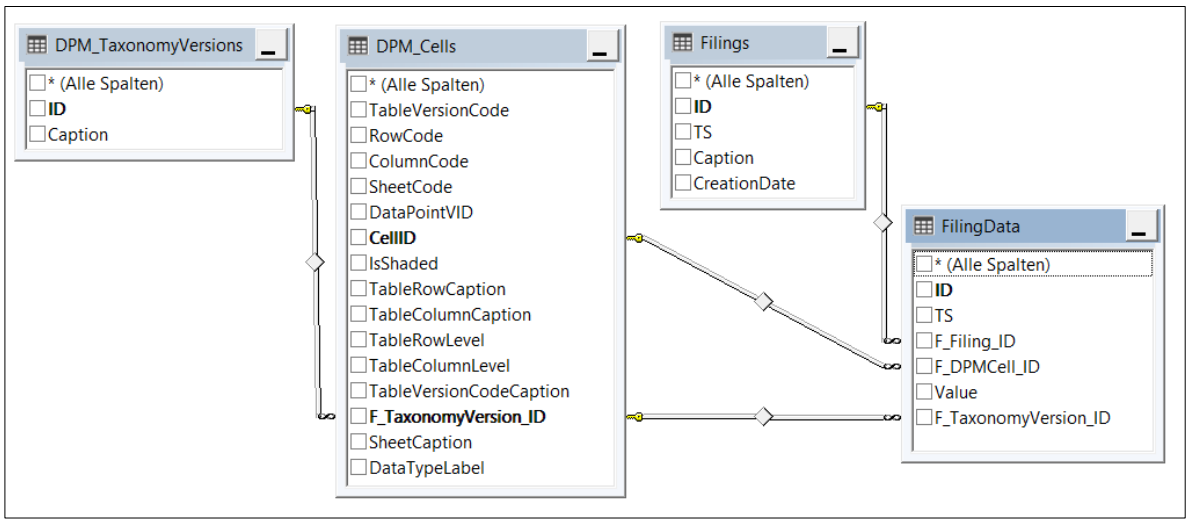


Abbildung 35: Teildatenmodell der Meldedaten

Quelle: eigenes Werk; Bildschirmfoto der gezeigten Tabellen der Parceval-Datenbank

Beim Validieren wird für alle Regeln der vom Anwender gewählten RuleCompilation folgender Ablauf ausgeführt. Die Ziffern in der Schilderung beziehen sich auf Abbildung 36, die das Verfahren an EBA-Regel v0835_m illustriert.

1. Alle Regeln der gewählten RuleCompilation werden ermittelt.
2. Zu den Regeln werden die Partitionen, Seiten, Vorzeichen, Faktoren und Zellbezüge ermittelt. So spannt sich ein Raster der Platzhalter auf, die die Formel anspricht.
3. Die Meldewerte (FilingData) des zu validierenden Laufs werden an den Zellbezügen eingetragen. Damit werden die Platzhalter belegt.
4. Wenn die Regel es verlangt, werden unbelegte Zellbezüge mit einem Standardwert belegt, etwa mit 0.
5. Die geforderten Berechnungen werden ausgeführt. Dabei entsteht in jedem Term ein numerischer Wert je Partition. Die Terme werden ggf. verrechnet. Es ergeben sich skalare Werte auf beiden Seiten des Vergleichs.
6. Der Vergleich ergibt einen Wahrheitswert je Partition. Ergeben alle Partitionen einer Regel den Wert wahr, ist die Regel erfüllt.
7. Das Endergebnis der Validierung ergibt aus den Ergebnissen *aller* Regeln. Sind keine Regeln verletzt, ist die Meldung valide. Bei verletzten Regeln entscheidet deren Schweregrad (severity), den die EBA für jede Regel vorgibt. Sind „blocking“-Regeln verletzt, sind die Daten nicht meldereif. Sind nur „non-blocking“-Regeln verletzt, ist eine Einreichung möglich, wird aber eine Erläuterung nötig machen oder Rückfragen nach sich ziehen. Im Beispielfall ist die „blocking“-Regel v0835_m verletzt, der gesamte Meldelauf damit unplausibel.

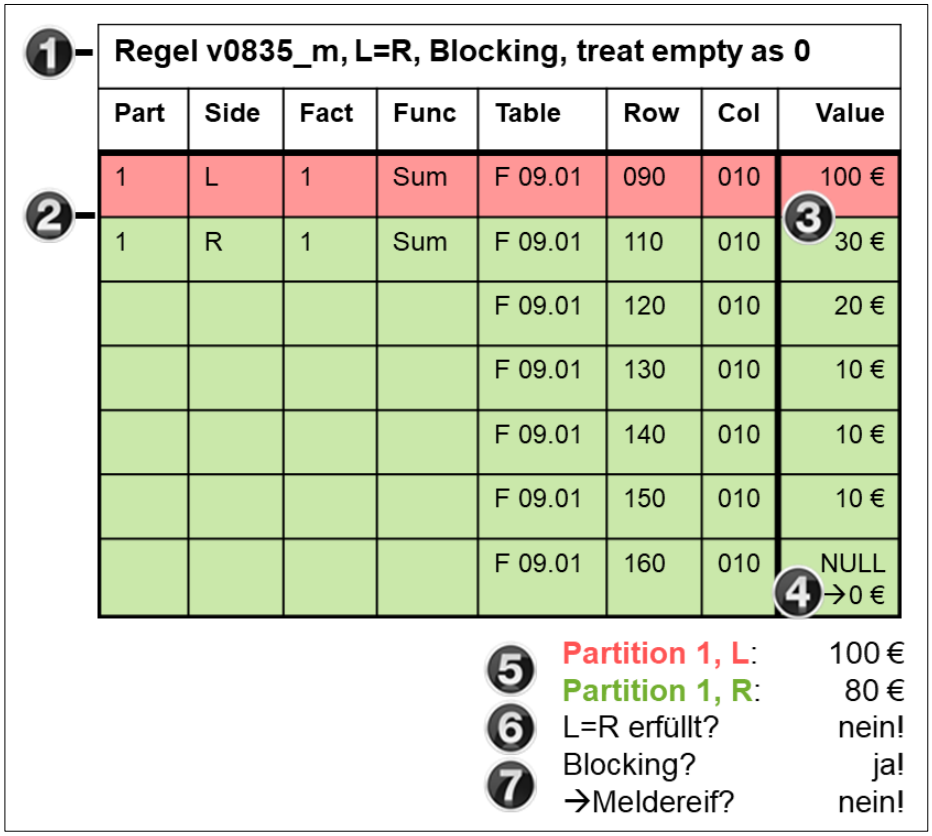


Abbildung 36: Validierung von Meldedaten

Quelle: eigenes Werk

In Parceval geschieht das Validieren eines Meldelaufs auf dem Rechner des Verfassers[20] in ca. 3 Sekunden. Das Gesamtergebnis für einen Meldestand aus Zufallszahlen ist – wenig überraschend – nicht valide, wie Abbildung 37 zeigt. Im Bild ist durch die aktuelle Auswahl in der linken Liste die Regel v0835_m gezeigt, diesmal mit Zufallswerten. Die drei Tabellen auf der rechten Seite zeigen von oben nach unten Herleitungsdetails. Die erste Tabelle deutet die geforderte, aber nicht erfüllte Gleichheit der Seiten als verletzte Regel (rötliche Füllung). Die zweite Tabelle rechts leitet die Seiten des Vergleichs aus seinen Termen her: die linke Seite des Vergleichs ist ein Term aus einer Einzelzelle, die rechte ein Term aus einer Summe mehrerer Zellen. Da der zweite Term markiert ist, erscheinen dessen Zellen in der dritten Tabelle. Schon diese verletzte Regel sorgt, da sie den Schweregrad „blocking“ hat, für die rote Ampel oben im Bild: Diese Daten sind nicht meldefähig. Die Statistik zeigt, dass auch weitere Regeln nicht erfüllt werden. In der linken Tabelle sind alle ausgeführten Regeln aufgeführt – erfüllte sind grün, verletzte rot gefärbt. Alle Listen sind kontextsensitiv, sie zeigen Details auf Mausklick.

[20] Vgl. Fußnote 17.

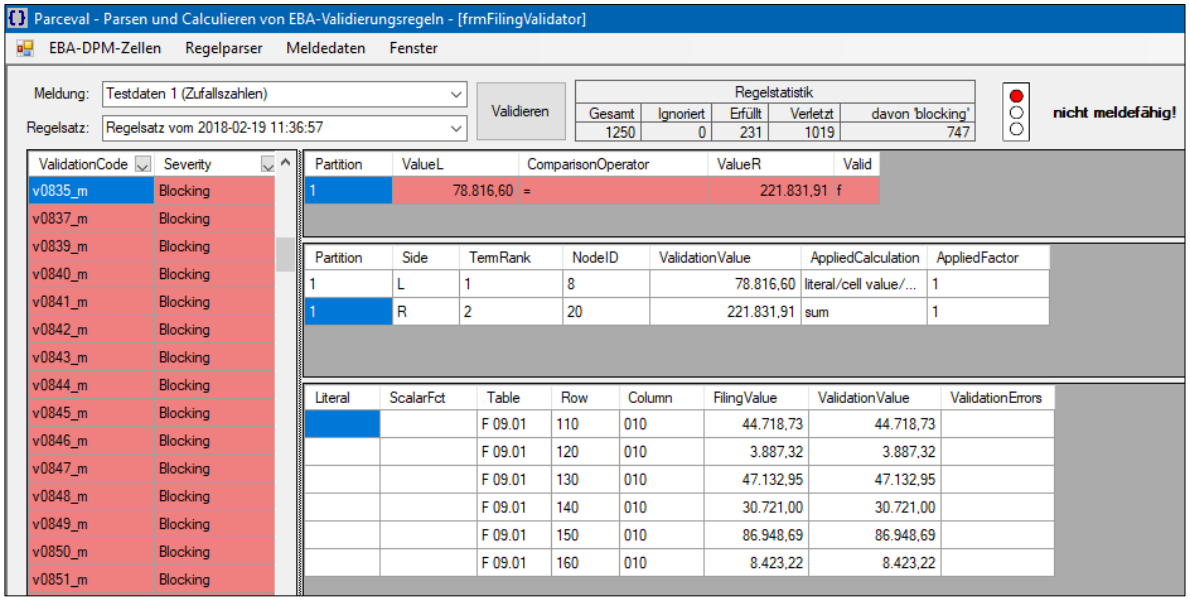


Abbildung 37: Validieren eines Meldestands in Parceval (Auszug)

Quelle: eigenes Werk; Bildschirmfoto aus Parceval

5 Ergebnis und Ausblick

Die These, dass sich die EBA-Validierungsregeln durch formale Grammatiken beschreiben lassen und sich folglich automatisch verarbeiten lassen, hat sich bestätigt.

Für die untersuchte Version 2.6 des EBA-Finrep-Meldewesens ist mit Parceval ein Anwendungsprototyp entstanden, der die 2849 Validierungsregeln parst, grammatisch korrekte Regeln in ihre Parsebäume zerlegt und diese bis auf wenige nicht unterstützte Regeln (s. u.) auf zu prüfende Meldedaten anwendet. Damit sind die Meldedaten schon vor Einreichung weitgehend auf ihre Validität prüfbar. Die Bank spart Zeit, vermeidet gewisse Reputationsschäden oder gar Sanktionen durch vermiedene Fehlmeldungen.

Sowohl Regeln als auch DPM-Zellen werden öffentlich zugänglichen Quellen entnommen. Künftige Änderungen an Zellvorrat oder Regelbestand können voraussichtlich ebenso nach Parceval übernommen werden. Die Bank kann schnell auf Änderungen reagieren und die Ableitungen aus den Zuliefersystemen ggf. anpassen.

Größerer Aufwand ist jedoch absehbar, falls sich Änderungen an den Grammatiken ergeben. In Version 2.7 des EBA-DPM haben sich bereits strukturelle Änderungen gegenüber der untersuchten Version 2.6 gezeigt, die auch für Parceval Folgen haben. Insbesondere werden seitdem die

Basistabellen nicht mehr nur als Spalten T1-T7 der EBA-Regeldatei veröffentlicht, sondern zusätzlich als boolescher Ausdruck in der Spalte **prerequisites** (Voraussetzungen) bedingt. So ist etwa Regel v5171_m in [EBA_VR_XLS_27] nur anzuwenden von Einreichern, die Werte in den Tabellen „F 18.00.a and (F 04.02.1 or F 04.02.2)“ melden[21]. Der Verfasser hat Parceval inzwischen an diese neue Schreibweise angepasst. Nun prüft das Programm, ob die Meldedaten Werte in den angesprochenen Tabellen enthalten, und setzt diese Wahrheitswerte in die prerequisites ein. Enthält eine Meldung z. B. Werte für F 18.00.a und F 04.02.1, nicht aber zu F 04.02.02, ergibt sich aus obigem prerequisites-Ausdruck „true and (true or false)“, was zu „true“ ausgewertet wird – die Regel ist für diese Meldung also anzuwenden, da die prerequisites erfüllt sind.

Die Entwicklung des DPM und der Validierungsregeln wird von der EBA weiter betrieben. In der Zeit zwischen Fertigstellung der Abschlussarbeit (März 2018) und dem Verfassen dieses Diskussionspapiers (Februar 2020) sind bereits die DPM-Versionen 2.7 bis 2.10 veröffentlicht worden [EBA_RFW], und DPM 3.0 zeichnet sich an gleicher Stelle bereits ab. Auch wenn der Ansatz von Parceval als Sprachlabor oder „language workbench“ [Völter_DSLE, S. 15] vielversprechend erscheint, lassen die häufigen und teils drastischen Veränderungen der EBA-Vorgaben einen hohen Pflegeaufwand auch in Zukunft erwarten.

Doch selbst für die untersuchte DPM-Version 2.6 ist Parceval ein Prototyp. Insbesondere folgende Mängel sollten in künftigen Entwicklungen behoben werden:

- Parceval unterstützt bislang nur die 3 Dimensionen Tabelle, Zeile, Spalte. In einigen Regeln kommt jedoch eine **4. Dimension namens Blatt (engl. sheet)** zum Einsatz, die Parceval bislang ausschließt. Im Gegensatz zum starren dreidimensionalen Modell sind die Blätter teilweise dynamisch. So fordert die EBA etwa in bestimmten Zellen des DPM die Detaillierung auf Herkunftsländer als Blatt. Um dünnbesetzte Tabellen zu vermeiden, werden dabei nicht alle ca. 200 Staaten der Welt erfragt, sondern nur diejenigen, die den Zellwert in der Meldung ausmachen. In der Abschlussarbeit wurde die 4. Dimension aus Zeitgründen ausgelassen. Dadurch werden 40 Regeln der untersuchten FinRep-Meldung als nicht unterstützt ausgeschlossen.
- Parceval schließt bisher einige Operationen aus, etwa **Bedingungen,**

[21] Die so abgedeckten Alternativen können z. B. durch Wahlrechte entstanden sein, die die EBA den Banken in ihren Bewertungen einräumt, oder sind Folge von Parametern wie Meldestichtag (Monats-/Quartals-/Jahresende) oder Rechnungslegungsart (HGB/IFRS).

Multiplikation oder Division. Auch hier war Komplexitätsreduzierung zur Zeitersparnis in der Abschlussarbeit der Grund. Allerdings ist diese Einschränkung im untersuchten Meldebereich nicht allzu schwerwiegend: 14 Regeln werden aus diesem Grund als nicht unterstützt ausgeschlossen.

- Ebenfalls ausgeschlossen sind **nichtnumerische Zellen** – Parceval kann bislang nur Zahlenwerte verarbeiten. In einigen Zellen werden jedoch andere Datentypen wie Datum oder Text erhoben. Diese Zellen und die Validierungsregeln, die sie einbeziehen, markiert Parceval als nicht verarbeitbar.

Parcevals Zukunft könnte in **Ausdrucksbäumen (expression trees)** liegen. Diese in C# nutzbaren „baumähnlichen Datenstrukturen“ [MS_ExpTree] kommen den abstrakten Syntaxbäumen sehr nah und lassen sich nach dem Baukastenprinzip oder aus Lambdaausdrücken zusammenstellen. Ausdrucksbäume können zahlreiche Operationen, die in den Validierungsregeln auftreten, direkt ausführen, etwa Bedingungen, Summen und Vergleiche. Sie können Parameter verarbeiten, die z. B. Dimensionskriterien aufnehmen könnten. Möglicherweise lassen sich Ausdrucksbäume automatisch aus den Parsebäumen aufbauen. In der Abschlussarbeit war dieser Ansatz nicht mehr umzusetzen – künftige Entwicklungen könnten aber auf dieser Idee basieren.

Außer Betracht blieb bislang auch die **Wandlung der Meldedaten in das XML-basierte XBRL-Format**, das für die Einreichung gefordert wird. Auch hier ist Potential für künftige Weiterentwicklung.

Ebenso ist die **Anlieferung von Produktivdaten** nicht implementiert, aber für einen Praxiseinsatz nötig. Hier sind Lieferstrecken zu Banksystemen einzurichten, etwa durch ETL[22]-Prozesse. Hier ist jedoch mit stark bankspezifischen Abläufen zu rechnen, die sich nach Ansicht des Verfassers kaum standardisieren lassen werden.

Parcevals Praxisnähe könnte weiter steigen, wenn die **Grunddaten der EBA** – DPM-Zellen und Validierungsregeln – künftig **automatisiert oder zumindest unterstützt eingelesen** werden könnten. Der bisherige manuelle Import war für den Rahmen einer Diplomarbeit akzeptabel, ist aber für einen zu erwartenden wiederkehrenden Geschäftsprozess untauglich. Allerdings könnte die EBA auch in Zukunft an DPM und Validierungsregeln nicht nur inhaltlich-fachliche, sondern auch technische Änderungen vornehmen. Die oben vorgestellten prerequisites der Regeln ab DPM v2.7 geben einen

[22] ETL: extract, transform, load.

Eindruck, wie gestaltungsfreudig die EBA nach wie vor ist. Ob Parceval mit dieser Dynamik mithalten kann, bleibt abzuwarten.

Mit diesem Diskussionspapier wird die Idee hinter Parceval veröffentlicht. Der Programmcode ist beim Verfasser erhältlich (Kontakt über den Herausgeber der WDP), die Diplomarbeit ist online verfügbar [Seip_DA] und natürlich an der Hochschule Wismar einsehbar. Der Verfasser sieht Rückfragen und Diskussionsbeiträgen zu diesem Papier gerne entgegen.

6 Literatur- und Internetquellen

[BrSc_BA]
Brixner, Joachim; Schaber, Mathias; *Bankenaufsicht: Institutionen, Regelungsbereiche und Prüfung*, Stuttgart, Schäffer Poeschel, 2016, ISBN 978-3-7910-3584-0.

[Cederberg_CLI]
Cederberg, Per; *Grammatica Reference Manual: Command-Line Interface,* https://grammatica.percederberg.net/doc/manual/commandline.html. Abgerufen am 18.02.2018.

[Cederberg_Grammatica]
Cederberg, Per; *Grammatica: a C# and Java parser generator (compiler compiler),* inkl. Download des Programms (grammatica-1.6.zip), https://grammatica.percederberg.net/. Abgerufen am 28.12.2017.

[EBA_AbsDescDPM]
EBA; *Abstract description of the model represented in taxonomies following the DPM approach*; Datei "Description of DPM formal model.pdf" in zip-Quelle [EBA_TaxDocZip].

[EBA_ATLDesc]
EBA; *Explanation for EBA CRR Annotated DPM Tables.pdf: Datei Explanation for EBA CRR Annotated DPM Tables.pdf in Quelle [EBA_DPMDocZIP]*.

[EBA_ATLZIP]
EBA; *DPM Table Layout and Data Point Categorisation.2.6.0.0.zip,* http://www.eba.europa.eu/documents/10180/1739088/DPM+Table+Layout+and+Data+Point+Categorisation.2.6.0.0.zip. Abgerufen am 18.02.2018.

[EBA_ATL_Finrep]
EBA; *Annotated Table Layout Finrep: Excel-Datei Annotated Table Layout 260-FINREP 2016-B.xlsx in zip-Quelle [EBA_ATLZIP]*.

[EBA_DPMDB_Desc_v2.0.0]
EBA; *CRD4 DPM - Database description - v2.0.0.pdf: in zip-Quelle [EBA_DPMDBv2.0.0ZIP]*.

[EBA_DPMDBv2.0.0ZIP]
EBA; *DPM Database.zip,* http://www.eba.europa.eu/documents/10180/502670/DPM+Database.zip. Abgerufen am 25.02.2018.

[EBA_DPMDBv2.6.0.0]
EBA; *DPM Database 2.6.0.0: Datei DPM Database 2.6.0.0.accdb in zip-Quelle [EBA_DPMDBv2.6.0.0ZIP]*.
Abgerufen am 18.02.2018.

[EBA_DPMDBv2.6.0.0ZIP]
EBA; *DPM Database.2.6.0.0 and 2.7.0.0.ED.zip*, http://www.eba.europa.eu/documents/10180/1739088/DPM+Database.2.6.0.0+and+2.7.0.0.ED.zip.
Abgerufen am 18.02.2018.

[EBA_DPMDocZIP]
EBA; *DPM Documents*, http://www.eba.europa.eu/documents/10180/1738725/DPM+Documents.zip.
Abgerufen am 03.03.2018.

[EBA_RFW]
EBA; *EBA reporting frameworks*, http://www.eba.europa.eu/risk-analysis-and-data/reporting-frameworks.
Abgerufen am 18.02.2018.

[EBA_RFW26]
EBA; *Reporting Framework 2.6*, https://www.eba.europa.eu/risk-analysis-and-data/reporting-frameworks/reporting-framework-2.6.
Abgerufen am 18.02.2018.

[EBA_TaxDocZIP]
EBA; *EBA Taxonomy and supporting documents.2.6.0.0.zip*, http://www.eba.europa.eu/documents/10180/1739095/EBA+Taxonomy+and+supporting+documents.2.6.0.0.zip.
Abgerufen am 18.02.2018.

[EBA_VR_Grammar]
EBA; *Validation formula.grammar: Datei in zip-Quelle [EBA_TaxDocZIP]*.

[EBA_VR_XLS]
EBA; *EBA Validation Rules: Excel-Datei EBA Validation Rules.xlsx, Tabellenblatt v2.6*, https://www.eba.europa.eu/documents/10180/1738006/EBA+Validation+Rules/3ca90745-406b-400f-b6cf-bf213d4ca644.
Abgerufen am 18.02.2018.

[EBA_VR_XLS_27]
EBA; *EBA Validation Rules: Excel-Datei EBA Validation Rules.xlsx, Tabellenblatt v2.7*, https://www.eba.europa.eu/documents/10180/1738006/EBA+Validation+Rules/3ca90745-406b-400f-b6cf-bf213d4ca644.
Abgerufen am 18.02.2018.

[Eberhardt_LTT]
Eberhardt, Colin; *LINQ to Tree - A Generic Technique for Querying Tree-like Structures*, https://www.codeproject.com/Articles/62397/LINQ-to-Tree-A-Generic-Technique-for-Querying-Tree#generic.
Abgerufen am 26.02.2018.

[EP_VO_575_2013]
Europäisches Parlament; Rat der Europäischen Union; *Verordnung (EU) Nr. 575/2013 des Europäischen Parlaments und des Rates über Aufsichtsanforderungen an Kreditinstitute und Wertpapierfirmen und zur Änderung der Verordnung (EU) Nr. 646/2012: "CRR"; VO (EU) Nr. 575/2013*, 26.06.2013, http://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32013R0575.
Abgerufen am 18.02.2018.

[EU_VO_1024_2013]
Rat der Europäischen Union; *Verordnung (EU) Nr. 1024/2013 des Rates vom 15. Oktober 2013 zur Übertragung besonderer Aufgaben im Zusammenhang mit der Aufsicht über Kreditinstitute auf die Europäische Zentralbank; VO (EU) Nr. 1024/2013*, 29.10.2013, https://eur-lex.europa.eu/legal-content/DE/ALL/?uri=celex:32013R1024.
Abgerufen am 01.03.2020.

[Lithner_FTV]
Lithner, Jakob; *Fast TreeView: Extension to the TreeView control making it very fast to load items*, 2013, https://www.codeproject.com/Articles/679563/Fast-TreeView.
Abgerufen am 18.02.2018.

[MS_ExpTree]
Microsoft; *Ausdrucksbaumstrukturen (C#)*, https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/concepts/expression-trees/.
Abgerufen am 27.02.2018.

[Parr_ANTLR]
Parr, Terence, *ANTLR - ANother Tool for Language Recognition*, https://www.antlr.org.
Abgerufen am 01.03.2020.

[Parr_LIP]
Parr, Terence; *Language implementation patterns: Create Your Own Domain-Specific and General Programming Languages*, 5. Auflage, Dallas, Raleigh, The Pragmatic Bookshelf, 2014, ISBN 978-1-934356-45-6.

[Seip_DA]
Seip, Martin, *Parsen und Evaluieren grammatikbasierter Validierungsregeln des Meldewesens der europäischen Bankenaufsicht*, Diplomarbeit an der Hochschule Wismar, 2018, https://www.fww.hs-wismar.de/storages/hs-wismar/_FWW/Professoren/Cleve/18-Diplom-Seip.pdf.
Abgerufen am 23.03.2020.

[Völter_DSLE]
Völter, Markus; *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*, 2013, http://voelter.de/dslbook/markusvoelter-dslengineering-1.0.pdf.
Abgerufen am 18.02.2017.

[VoWi_GTI]
Vossen, Gottfried; Witt, Kurt-Ulrich; *Grundkurs Theoretische Informatik: Eine anwendungsbezogene Einführung - für Studierende in allen Informatik-Studiengängen,* 6. Auflage, Wiesbaden, Springer Vieweg, 2016, ISBN 978-3-8348-1770-9.

[WaHi_FSAAC]
Wagenknecht, Christian; Hielscher, Michael; *Formale Sprachen, abstrakte Automaten und Compiler: Lehr- und Arbeitsbuch für Grundstudium und Fortbildung,* 2. Auflage, Wiesbaden, Springer Vieweg, 2014, ISBN 978-3-658-02691-2.

WDP - Wismarer Diskussionspapiere / Wismar Discussion Papers

Heft 04/2018: Claudia Walden-Bergmann: Nutzen und Nutzung von E-Learning-Angeboten im Präsenzstudium Analyse von Daten des Moduls Investition

Heft 05/2018: Sonderheft: Katrin Schmallowsky, Christian Feuerhake, Empirische Studie zum Messeverhalten von kleinen und mittleren Unternehmen in Mecklenburg-Vorpommern

Heft 06/2018: Dieter Gerdesmeier, Barbara Roffia, Hans-Eggert Reimers: Unravelling the secrets of euro area inflation – a frequency decomposition approach

Heft 07/2018: Harald Mumm: Didaktischer Zugang zur Theorie und Praxis moderner Softwarebibliotheken (Frameworks) für die Unternehmensforschung (OR)

Heft 01/2019: Astrid Massow: Deutsche Bank AG und Commerzbank AG – Neubewertung der Unternehmen im Rahmen einer potenziellen Bankenfusion

Heft 02/2019: Günther Ringle: Das genossenschaftliche Identitätsprinzip: Anspruch und Wirklichkeit

Heft 01/2020: Luisa Lore Ahlers: Einführung eines Wissensmanagements in kleinen und mittleren Unternehmen am Beispiel der Stadtwerke Wismar GmbH

Heft 02/2020: Harald Mumm: Didaktischer Zugang zur Theorie und Praxis moderner Softwarebibliotheken (Frameworks) für die Unternehmensforschung (OR)

ISBN 978-3-948862-00-8